

软件工程师典藏



Visual C++

程序开发

范例宝典

明日科技 曹飞飞 赵永发 吴绪铎 编著



- 400个典型范例，315分钟范例视频讲解
- 知识点全面，涵盖Visual C++程序开发中用到的各种技术
- www.mrbccd.com 12小时最新免费学习课程
- 每周5天，每天9小时答疑服务



- 书中所有范例的源代码
- 20小时多媒体语音视频教学录像
- 明日科技免费赠送的软件产品



人民邮电出版社
POSTS & TELECOM PRESS

范例宝典，经典难以复制！

程序开发范例宝典

实例说明

以图文结合的形式给出实例功能和运行效果

技术要点

给出了实例开发过程中的重点、难点技术

实现过程

介绍实例的设计过程和主要程序代码

举一反三

读者可以根据所学扩展应用

窗体与控件

窗体与界面设计

控件应用

图形与多媒体

图形技术

多媒体技术

文件、系统与注册表

文件系统

操作系统与Windows相关程序

注册表

数据库技术

数据库技术

SQL查询相关技术

打印与报表技术

网络开发

网络开发技术

Web编程

数据安全

加密、安全与软件注册

硬件相关开发技术

实用工具开发

实用工具

全程跟踪服务

如果您在使用本书时遇到什么困难或疑惑，5个工作日内将得到解答
周一至周五8:00~17:00全程服务，为您解答编程难题
QQ、TQ、电话、邮件，立体服务，让您编程畅通无阻



分类建议：计算机 / 程序设计 / Visual C++
人民邮电出版社网址：www.ptpress.com.cn

封面设计：任文杰

ISBN 978-7-115-27795-4



9 787115 277954 >

ISBN 978-7-115-27795-4

定价：98.00 元（附光盘）



Visual C++

程序开发

范例宝典

明日科技 曹飞飞 赵永发 吴绪铎 编著

人民邮电出版社
北京

图书在版编目 (CIP) 数据

Visual C++程序开发范例宝典 / 曹飞飞, 赵永发,
吴绪铎编著. — 北京: 人民邮电出版社, 2012. 5
ISBN 978-7-115-27795-4

I. ①V… II. ①曹… ②赵… ③吴… III. ①
C语言—程序设计 IV. ①TP312

中国版本图书馆CIP数据核字 (2012) 第047199号

内 容 提 要

本书以开发人员在编程中遇到的实际问题和开发中应该掌握的技术为中心, 全面介绍运用 Visual C++ 进行程序开发的各方面技术和技巧。全书包括窗体与界面设计, 控件应用, 图形技术, 多媒体技术, 文件系统, 操作系统与 Windows 相关程序, 注册表, 数据库技术, SQL 查询相关技术, 打印与报表技术, 硬件相关开发技术, 网络开发技术, Web 编程, 加密、安全与软件注册, 实用工具等共 15 章, 共 400 个实例, 每个实例都突出实用性, 其中大部分是程序开发者梦寐以求的问题解决方案。

本书附有配套光盘。光盘提供了书中所有实例的源代码, 所有代码都经过精心调试, 在 Windows XP/Windows 2000 下测试通过, 均能正常运行。

本书适合 Visual C++ 编程人员阅读使用, 也可供大中专院校师生学习参考。

Visual C++程序开发范例宝典 (第3版)

- ◆ 编 著 明日科技 曹飞飞 赵永发 吴绪铎
责任编辑 蒋 佳
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
三河市海波印务有限公司印刷
- ◆ 开本: 787×1092 1/16
印张: 49.5
字数: 1 340 千字 2012 年 5 月第 3 版
印数: 14 001—17 000 册 2012 年 5 月河北第 1 次印刷

ISBN 978-7-115-27795-4

定价: 98.00 元 (附光盘)

读者服务热线: (010) 67132692 印装质量热线: (010) 67129223

反盗版热线: (010) 67171154

广告经营许可证: 京崇工商广字第 0021 号

前言

几年前,笔者参加了一个项目的开发工作,项目要求时间很紧,开发团队几乎是挑灯夜战。当时基于 Visual C++ 的开发资料很少,网络也不发达,常常为了解决一个问题,大家连续奋战几天、十几天,甚至几十天。之后,笔者又参加了多个项目的开发工作。在开发过程中深刻地感觉到:编程是一项创造性较强的工作,因其涉及面广,开发者往往需要学习、研究各方面的技术和问题;编程水平的提高与开发时间成正比,需要长时间经验的积累和磨炼;编程也是一项需要相互学习、相互交流的工作,在交流过程中,不但可以分享他人的编程经验、体会,更会产生创新的灵感,达到事半功倍的效果。

总之,项目开发不是一件容易的事,即使是非常有经验的开发人员,也经常会遇到一些开发技术难题,要成为一个合格的程序员,就必须不断吸取和借鉴其他开发者的成功经验。通过阅读别人的程序,从中吸取编程思想的精华,这是学习程序设计最好的方法。

■ 本书内容 ■

本书精选了 400 个典型实例,所选实例主要覆盖开发中的热点问题和关键问题。全书按实际应用进行分类,可以使读者在短时间内掌握更多有用的技术,快速提高编程水平。所选内容均来源于实际项目的开发,有的实例是作者开发实践的积累,有的实例来源于公司的开发项目,还有的来自读者的问题。这些实例所涉及的问题都是非常实用的。

通过对这些实例进行详细分析和讲解,可以让读者迅速掌握程序设计的开发经验技巧,迅速提高程序设计的综合水平。全书分为 15 章,涵盖了 Visual C++ 的窗体与界面设计,控件应用,图形技术,多媒体技术,文件系统,操作系统与 Windows 相关程序,注册表,数据库技术,SQL 查询相关技术,打印与报表技术,硬件相关开发技术,网络开发技术,Web 编程,加密、安全与软件注册,实用工具等方面的内容。

在实例讲解上,全书采用了统一的编排方式,每个实例都包括“实例说明”、“技术要点”、“实现过程”和“举一反三”4 个部分。在“实例说明”中,以图文结合的方式给出了实例的功能说明及运行效果。在“技术要点”中给出了实例的重点、难点技术和相关编程技巧。在“实现过程”中介绍了该实例的设计过程和主要程序代码。在“举一反三”中给出了相关实例的扩展应用。

■ 本书特色 ■

- 所有实例内容都以解决开发者在编程中遇到的实际问题和开发中应该掌握的技术为中心,每个实例都可以独立解决某一方面的问题。有的可以解决工作中的难题,有的可以提高工作效率,有的可以提升工作价值。
- 所选实例具有极强的扩展性,能够给读者以启发,使读者举一反三,开发出非常实用的应用程序。
- 所选实例具有广泛的代表性,所有实例都提供了源代码,方便读者使用。
- 为便于查找实例中的编程技术和技巧,本书附录提供了程序快速索引功能,该索引按字母顺序列出了本书中使用的相关技术和技巧的实例号。

- 本书附带光盘提供了本书实例的“源码速查电子搜索引擎”，读者可以快速检索到所需的技术和代码。

■ 本书的约定 ■

- 书中每个实例的标题栏都给出了程序的特色和所在光盘中的路径，读者可以根据需要学习和使用。
- 书中涉及数据库的实例，在实例对应文件夹中均提供了数据库文件或数据库文件路径。
- 书中可能多个实例用到了同一主要技术，为节省篇幅，相关技术只在一个实例中进行介绍，读者可通过书后的技术索引了解相关技术的章节位置。
- 因篇幅限制，书中实例只给出了关键代码，其他代码参见光盘中实例的源程序。
- 使用本书实例光盘前，请仔细阅读光盘中的“光盘使用说明”。

■ 第3版所做的改进 ■

在第3版中，我们主要遵循以下原则对第2版内容进行修改。

- 改进了内容

增加了 Visual C++ 程序开发相关的各种新技术和热点应用，使本书更贴近实际开发应用。如新增了控件自绘、图形图像等多方面的实例，将原书中的不常用的实例加以替换。

- 增强了易懂性

在第2版中有一些内容的阐述或说明比较难理解，不便于读者掌握，在第3版中我们更换了一些图片，修改了不利于理解的文字。

■ 本书的服务 ■

本书由明日科技组织编写，参加编写的有曹飞飞、赵永发、吴绪铎、王国辉、朱晓、刘欣、李伟、杨丽、高文才、王小科、李银龙、顾彦玲、赵会东、潘凯华、李继业、陈英、孙茜、寇长梅、陈丹丹、李慧等。书中疏漏和错误之处在所难免，敬请广大读者批评指正。

为便于读者和本书作者沟通，明日科技将通过明日科技网站全面为读者提供网上服务和支持。读者使用本书遇到的错误和问题，我们承诺在5个工作日内给您提供及时答复。

服务网站：www.mingribook.com

服务信箱：mingrisoft@mingrisoft.com

客服电话：0431-84978981 84978982

本书编写组

2012年3月

目 录

第1章 窗体与界面设计 1

1.1 菜单应用实例 2

实例 001 在系统菜单中添加
菜单项 2

实例 002 带图标的程序菜单 3

实例 003 根据 INI 文件创建菜单 6

实例 004 浮动的菜单 7

1.2 弹出菜单应用实例 9

实例 005 在控件上单击右键弹出
菜单 9

实例 006 个性化的弹出菜单 9

实例 007 任务栏托盘弹出菜单 12

1.3 工具栏应用实例 14

实例 008 根据菜单创建工具栏 14

实例 009 带图标的工具栏 15

实例 010 定制浮动工具栏 16

实例 011 可调整按钮位置的
工具栏 18实例 012 在工具栏中添加
编辑框 19实例 013 动态设置是否显示工具
栏按钮文本 20实例 014 具有提示功能的
工具栏 21

1.4 状态栏应用实例 23

实例 015 使状态栏随对话框的
改变而改变 23

实例 016 动画效果的状态栏 24

实例 017 滚动字幕的状态栏 25

1.5 导航界面应用实例 26

实例 018 Outlook 导航界面 26

实例 019 树状导航界面 28

实例 020 按钮导航界面 30

实例 021 图片导航界面 32

1.6 界面窗体应用实例 33

实例 022 使用位图设计畸形界面 33

实例 023 制作立体窗口阴影效果 35

实例 024 自绘窗体界面 36

实例 025 以时钟显示界面 41

实例 026 窗体融合技术 42

实例 027 限制对话框最大时的
窗口大小 46

实例 028 分割视图窗口 47

实例 029 Animate 动画显示窗体 48

1.7 多媒体宣传光盘应用实例 49

实例 030 多媒体宣传光盘
主界面 49实例 031 自动运行的多媒体宣传
光盘 50

1.8 多媒体触摸屏程序应用实例 51

实例 032 采购中心多媒体触摸屏
程序 51实例 033 为触摸屏程序添加虚拟
键盘 53

1.9 窗体位置应用实例 54

实例 034 不可移动的窗体 54

实例 035 始终在最上面的窗体 54

实例 036 如 QQ 般隐藏的窗体 55

实例 037 磁性窗体 57

1.10 窗体标题栏应用实例 58

实例 038 闪烁的窗体标题栏 58

实例 039 隐藏和显示标题栏 59

实例 040 禁用标题栏上的最大化、
最小化或关闭按钮 59

1.11 窗体形状及应用 61

实例 041 半透明窗体 62

实例 042 创建字型窗体 63

实例 043 换肤窗体 64

1.12 通用对话框的应用 66

实例 044 打开位图预览对话框 66

实例 045 打开 Windows 新型
对话框 68

实例 046 同时选择多个文件 69

实例 047 文本替换对话框 70

实例 048 字体选择对话框 72

第2章 控件应用 75

- 2.1 按钮控件典型实例 76
 - 实例 049 AVI 动画按钮 76
 - 实例 050 GIF 动画按钮 78
 - 实例 051 图文按钮 80
 - 实例 052 按钮七巧板 82
 - 实例 053 热点按钮 84
- 2.2 编辑框控件典型实例 86
 - 实例 054 为编辑框设置新的系统菜单 87
 - 实例 055 为编辑框控件添加列表选择框 88
 - 实例 056 多彩边框的编辑框 90
 - 实例 057 改变编辑框文本颜色 91
 - 实例 058 不同文本颜色的编辑框 92
 - 实例 059 位图背景编辑框 93
- 2.3 静态文本控件典型实例 94
 - 实例 060 电子时钟 94
 - 实例 061 文本背景的透明处理 96
 - 实例 062 制作超链接控件 97
- 2.4 列表框控件典型实例 99
 - 实例 063 利用列表框控件实现标签式数据选择 99
 - 实例 064 以报表显示图书信息 100
 - 实例 065 QQ 抽屉界面 102
 - 实例 066 位图背景列表框控件 103
- 2.5 组合框控件典型实例 105
 - 实例 067 将数据表中的字段添加到组合框控件 105
 - 实例 068 带查询功能的组合框控件 106
 - 实例 069 自动调整组合框的宽度 108
 - 实例 070 颜色组合框 109
 - 实例 071 多列显示的组合框 111
 - 实例 072 QQ 登录式的用户选择列表 113
 - 实例 073 显示系统盘符组合框 114
- 2.6 列表视图控件典型实例 115
 - 实例 074 Windows 资源管理器 115

- 实例 075 利用列表视图控件浏览数据 118
- 实例 076 利用列表视图控件制作导航界面 119
- 实例 077 在列表视图中拖动视图项 121
- 实例 078 具有排序功能的列表视图控件 122
- 实例 079 具有文本录入功能的列表视图控件 125
- 实例 080 使用列表视图设计登录界面 127
- 2.7 树视图控件典型实例 129
 - 实例 081 多级数据库树状结构数据显示 129
 - 实例 082 节点拖动功能的树控件 131
 - 实例 083 带复选功能的树状结构 134
 - 实例 084 三态效果树控件 135
 - 实例 085 修改树控件节点连线颜色 137
 - 实例 086 位图背景树控件 138
 - 实例 087 显示磁盘目录 139
 - 实例 088 树型提示框 141
- 2.8 RichEdit 控件典型实例 143
 - 实例 089 利用 RichEdit 显示 Word 文档 143
 - 实例 090 利用 RichEdit 控件实现文字定位与标识 144
 - 实例 091 利用 RichEdit 控件显示图文数据 145
 - 实例 092 在 RichEdit 中显示不同字体和颜色的文本 147
 - 实例 093 在 RichEdit 中显示 GIF 动画 149
- 2.9 滚动条控件典型实例 157
 - 实例 094 自定义滚动条控件 157
- 2.10 进度条控件典型实例 162
 - 实例 095 进度条百分比显示 162
 - 实例 096 渐变颜色的进度条 163
- 2.11 工具提示控件典型实例 165

实例 097 应用工具提示控件	165	实例 126 图片的平滑缩放	220
2.12 滑块控件典型实例	166	3.6 图像的剪切、合成与识别	222
实例 098 使用滑块控件设置 颜色值	166	实例 127 图像的剪切	222
实例 099 绘制滑块控件	168	实例 128 图像的合成	224
2.13 标签控件典型实例	170	实例 129 获取鼠标任意位置的 颜色值	225
实例 100 应用标签控件	170	实例 130 提取图片中的对象	226
实例 101 自定义标签控件	171	实例 131 手写数字识别	228
2.14 控件数组典型实例	175	3.7 图像字体	231
实例 102 向窗体中动态添加 控件	175	实例 132 旋转的文字	231
实例 103 公交线路模拟	177	实例 133 当前系统字体列表	233
第 3 章 图形技术		实例 134 空心文字	234
179		实例 135 彩虹文字	235
3.1 绘制图形	180	实例 136 如何在图片上平滑移动 文字	236
实例 104 绘制正弦曲线	180	实例 137 图像水印效果	238
实例 105 绘制蜗牛曲线	181	3.8 图像管理	240
实例 106 绘制贝塞尔曲线	182	实例 138 管理计算机内图片 文件的程序	240
实例 107 画图程序	183	实例 139 提取并保存应用程序 图标	243
实例 108 绘制立体模型	184	3.9 图片动画	245
实例 109 利用 IFS 算法绘制自然 景物	186	实例 140 利用图片制作屏幕保护 程序	245
3.2 图像预览	188	实例 141 图片动画	246
实例 110 图片自动预览程序	188	实例 142 指法练习软件	247
实例 111 图片批量浏览	189	3.10 简单游戏设计	251
实例 112 浏览大幅 BMP 图片	192	实例 143 拼图游戏	251
实例 113 放大和缩小图片	195	实例 144 黑白棋	255
实例 114 图像任意角度旋转	197	实例 145 俄罗斯方块	257
3.3 图片效果	204	实例 146 快来打地鼠	259
实例 115 图片马赛克效果	204	实例 147 幸运转盘	260
实例 116 图片百叶窗效果	207	3.11 OpenGL 程序设计	262
实例 117 电影胶片特效	209	实例 148 制作 OpenGL 动画	262
实例 118 翻转图片效果	210	实例 149 利用 OpenGL 绘制立体 模型	265
实例 119 图片浮雕效果	212	实例 150 利用 OpenGL 绘制 NURBS 曲线	268
3.4 图片颜色转换	213	3.12 GDI+程序设计	270
实例 120 图像的锐化处理	213	实例 151 使用 GDI+显示 GIF 动画	270
实例 121 图片反色处理	215	实例 152 使用 GDI+实现图像	
实例 122 图像的灰度化转换	216		
实例 123 显示 JPG 图片	217		
3.5 图形转换与缩放	219		
实例 124 将位图转换为 JPG	219		
实例 125 将位图转为 GIF 图标	220		

第4章 多媒体技术 275

4.1 动画 276	实例 153 屏幕动画精灵 276	实例 154 利用位图制作 AVI 动画 278	实例 155 播放 GIF 动画 281	实例 156 播放 Flash 动画 282	实例 157 文字跟随鼠标 283
4.2 制作与播放音频 284	实例 158 可以选择播放曲目的 CD 播放器 284	实例 159 开发具有记忆功能的 MP3 播放器 286	实例 160 声音录制与播放 287	实例 161 制作 RealOne 播放器 288	
4.3 多媒体控制 290	实例 162 音频波形显示 290	实例 163 利用 PC 喇叭播放声音 292	实例 164 控制左右声道 294		
4.4 屏幕保护相关程序 295	实例 165 电子相册屏幕保护程序 295	实例 166 产品宣传屏幕保护程序 297	实例 167 滚动字幕屏幕保护程序 299		
4.5 DirectShow 程序设计 300	实例 168 音频捕捉 300	实例 169 音频压缩 304	实例 170 视频捕捉 308	实例 171 视频压缩 310	实例 172 使用 DirectShow 设计媒体播放器 316

第5章 文件系统 331

5.1 文件的基本操作 332	实例 173 创建和删除文件夹 332	实例 174 把文件删除到回收站中 333
-----------------------	---------------------------	-----------------------------

实例 175 清空回收站 334	实例 176 强制删除文件 335	5.2 查找文件 340	实例 177 搜索文件 340	实例 178 使用多线程实现文件快速搜索 342	实例 179 检查文件是否存在 344	实例 180 提取指定文件夹目录到 INI 文件 345	5.3 与文件目录相关的命令操作 347	实例 181 删除文件目录 347	实例 182 重命名文件目录 348	5.4 文件、文件夹的复制和移动 349	实例 183 批量移动文件 349	实例 184 网络文件夹复制 351	实例 185 文件复制过程中显示进度条 353	5.5 文件修改 355	实例 186 修改应用程序图标 355	实例 187 更改文件夹图标 358	实例 188 批量删除指定类型的文件 360	实例 189 批量重命名文件 361	实例 190 修改文件属性 363	实例 191 修改文件及目录的名称 365	5.6 文件的读取与保存 367	实例 192 顺序读取文件 368	实例 193 制作日志文件 369	实例 194 获取 Word 文档属性 370	实例 195 将 Word 转换为 HTML 373	实例 196 提取 Word 文档目录 374	5.7 文件管理 376	实例 197 分类整理磁盘文件 376	实例 198 计算机磁盘空间报警程序 378	实例 199 批量改变指定文件的属性 380	5.8 加密与解密 382	实例 200 文件的加密与解密 382	实例 201 文件夹加密 384
------------------------	-------------------------	--------------------	-----------------------	--------------------------------	---------------------------	------------------------------------	----------------------------	-------------------------	--------------------------	----------------------------	-------------------------	--------------------------	-------------------------------	--------------------	---------------------------	--------------------------	------------------------------	--------------------------	-------------------------	-----------------------------	------------------------	-------------------------	-------------------------	-------------------------------	----------------------------------	-------------------------------	--------------------	---------------------------	------------------------------	------------------------------	---------------------	---------------------------	------------------------

5.9	INI 文件	385
实例 202	向 INI 文件中写入数据	385
实例 203	使用 INI 文件保存配置信息	386
5.10	其他	388
实例 204	文件分割器	388
实例 205	用 WinRar 压缩和解压文件	390
实例 206	捆绑可执行文件	392
实例 207	读写 XML 文件	395

第 6 章 操作系统与 Windows 相关程序 397

6.1	启动相关	398
实例 208	进入 WinXP 前发出警告	398
实例 209	实现关机、重启计算机	399
实例 210	将程序设置成为开机自动执行的程序	400
6.2	磁盘相关	401
实例 211	判断驱动器属性	401
实例 212	获取磁盘空间信息	403
实例 213	获取磁盘序列号	404
实例 214	取消磁盘共享	405
实例 215	格式化磁盘	406
6.3	桌面相关	408
实例 216	隐藏、显示开始按钮	408
实例 217	隐藏、显示桌面文件	409
实例 218	隐藏、显示 Windows 任务栏	410
实例 219	随机修改系统桌面背景	411
实例 220	抓取桌面	413
6.4	系统相关	417
实例 221	获得 Windows 和 System 的路径	417
实例 222	控制光驱的弹开与关闭	418
实例 223	启动控制面板	419
实例 224	定时关闭计算机	421
实例 225	实现 OCX 控件的注册和卸载	425
6.5	系统监控	427
实例 226	检测 U 盘是否插入	427
实例 227	检测文件和目录是否改变	429
实例 228	检测系统启动模式	432
实例 229	内存使用状态	433
实例 230	监视剪贴板内容	434
实例 231	利用钩子技术实现键盘监控	435
6.6	程序相关	437
实例 232	用列表显示系统正在运行的程序	437
实例 233	为程序添加快捷方式	438
实例 234	设置其他程序中编辑框内的文本	440
实例 235	执行一个外部程序直到其结束	441
实例 236	调用具有参数的可执行程序	443
实例 237	编写控制面板小应用程序	445
实例 238	编写 Windows 服务	446
实例 239	阻止程序重复运行	449
6.7	线程同步	450
实例 240	利用事件对象实现线程同步	450
实例 241	利用互斥对象实现线程同步	452
实例 242	利用临界区实现线程同步	453
实例 243	用信号量实现线程同步	454
实例 244	多线程实例	456
6.8	鼠标、键盘相关	458
实例 245	动画鼠标	458
实例 246	限制鼠标移动区域	459
实例 247	鼠标穿透窗体	460
实例 248	设置鼠标形状	462
实例 249	控制键盘指示灯	463
6.9	动态链接库	464

- 实例 250 访问 DLL 中的位图464
实例 251 从 DLL 中导出类对象465

第7章 注册表467

- 7.1 显示与隐藏468
实例 252 隐藏、显示“我的电脑”、“回收站”、“网上邻居”468
实例 253 隐藏、显示驱动器470
7.2 IE 浏览器设置471
实例 254 修改 IE 浏览器标题栏内容471
实例 255 隐藏 IE 浏览器的右键关联菜单472
实例 256 设置 IE 浏览器的默认主页473
实例 257 清空上网历史记录474
7.3 文件控制475
实例 258 如何建立文件关联475
实例 259 控制光驱的自动运行功能477
7.4 游戏设置478
实例 260 设置“蜘蛛纸牌”游戏478
实例 261 修改“扫雷”游戏的设置480
7.5 应用软件设置481
实例 262 设置 Word 2000 文档及图片的保存路径482
实例 263 更改 Photoshop 安装时的登记信息483

第8章 数据库技术485

- 8.1 连接数据库486
实例 264 使用 ODBC DSN 连接 SQL Server 数据库486
实例 265 用 ADO 动态连接数据库488
8.2 添加数据491
实例 266 利用 INSERT 语句批量插入数据491
实例 267 利用 SELECT INTO 生成临时表492

- 8.3 更新数据493
实例 268 批量修改数据494
实例 269 将指定字段数据为空的记录添上数据495
8.4 删除数据495
实例 270 删除单条数据496
实例 271 删除数据库中无用处的记录496
8.5 视图497
实例 272 动态创建视图498
实例 273 通过视图更改数据499
实例 274 删除视图499
8.6 存储过程500
实例 275 创建存储过程500
实例 276 删除存储过程502
实例 277 在程序中使用存储过程503
实例 278 调用具有输出参数的存储过程504
实例 279 编写扩展存储过程505
8.7 数据库结构的读取与修改507
实例 280 读取 Access 数据库结构507
实例 281 读取 SQL Server 数据库结构509
8.8 图片、多媒体数据录入技术510
实例 282 对 Access 数据库进行录入和提取图片510
实例 283 对 SQL Server 数据库进行录入和提取多媒体文件513
8.9 数据备份恢复515
实例 284 Access 数据库备份与还原515
实例 285 SQL Server 数据库备份与恢复516
实例 286 定时数据备份519
8.10 其他数据库技术520
实例 287 断开 SQL Server 数据库与其他应用程序的连接520
实例 288 在 Visual C++ 中执行

事务	521
实例 289 在程序中执行 SQL 脚本	522
实例 290 利用 SQL 语句执行 外国命令	524
实例 291 枚举 SQL Server 服务器	524
实例 292 附加数据库	526
实例 293 分离数据库	527

第 9 章 SQL 查询相关技术 529

9.1 通用查询	530
实例 294 SELECT 语句的应用 方法	530
实例 295 SQL 语句的模糊查询	531
实例 296 利用查询语句复制表 结构	532
9.2 周期、日期查询	533
实例 297 查询指定时间段的 数据	534
实例 298 按月查询数据	535
实例 299 在查询中使用日期 函数	536
9.3 比较、逻辑、重复记录查询	537
实例 300 NOT 与谓词进行组合 条件的查询	537
实例 301 查询时不显示重复 记录	538
9.4 排序、分组统计	539
实例 302 对数据进行降序查询	540
实例 303 对数据进行多条件 排序	541
9.5 聚集函数	542
实例 304 利用聚集函数 SUM 对 销售额进行汇总	542
实例 305 利用聚集函数 AVG 求 某班学生的平均年龄	543
实例 306 利用聚集函数 COUNT 求日销 售额大于某值的商品数	544

第 10 章 打印与报表技术 545

10.1 基础打印	546
-----------------	-----

实例 307 基于文档/视图结构的 打印	546
实例 308 基于对话框结构的打印 程序	549
实例 309 打印对话框及其控件中的 数据	550
10.2 打印图片	552
实例 310 打印图片	552
实例 311 打印简历	553
10.3 打印单据	558
实例 312 打印汇款单	558
实例 313 打印信封标签	561
实例 314 假条套打	563
实例 315 批量打印条形码	564
10.4 控制打印	567
实例 316 批量打印文档	567
实例 317 实现横向打印	568
实例 318 设置打印表格的边线及 字体	570
10.5 打印预览	572
实例 319 具有滚动条的预览 界面	572
实例 320 在对话框中分页预览	577

第 11 章 硬件相关开发技术 583

11.1 串口控制	584
实例 321 通过串口传递数据	584
实例 322 通过串口控制对方 计算机关闭	588
11.2 加密狗和加密锁	590
实例 323 将密码写入加密狗	590
实例 324 使用加密狗进行身份 验证	591
实例 325 将数据写入加密锁	592
实例 326 使用加密锁进行软件 注册	593
11.3 IC 卡、ID 卡应用	595
实例 327 向 IC 卡中写入数据	595
实例 328 读取 IC 卡中的数据	598
实例 329 利用 IC 卡制作考勤 程序	599
实例 330 使用 ID 卡制作考勤	

程序.....	601	12.2 局域网控制与管理.....	653
11.4 监控.....	604	实例 354 获取局域网计算机 名称和 IP.....	653
实例 331 利用简易摄像头编写 监控程序.....	604	实例 355 远程控制局域网 计算机.....	654
实例 332 编写监控录像程序.....	606	12.3 局域网资源管理.....	657
实例 333 远程视频监控系统.....	607	实例 356 计算机监控.....	657
实例 334 云台控制.....	610	实例 357 实现进程间通信.....	660
11.5 扫描、条形码、POS 控制.....	614	实例 358 利用内存映射实现进程 间通信.....	662
实例 335 利用条形码扫描器销售 商品.....	614	12.4 网上资源共享.....	663
实例 336 使用数据采集器进行库 存盘点.....	617	实例 359 获得网上共享资源.....	664
实例 337 设计钱箱控制程序.....	618	实例 360 映射网络驱动器.....	665
实例 338 设计扫描仪控制程序.....	620	12.5 套接字应用.....	666
实例 339 设计发票机控制程序.....	620	实例 361 网络聊天室.....	666
11.6 语音卡控制.....	621	实例 362 语音实时通信.....	669
实例 340 语音卡电话呼叫系统.....	621	实例 363 视频聊天室.....	672
实例 341 语音卡实现来电显示.....	626	12.6 其他.....	677
实例 342 利用语音卡实现电话 录音.....	629	实例 364 获得拨号网络的列表.....	677
实例 343 利用语音卡实现点歌 祝福.....	631	实例 365 获取计算机上串口的 数量.....	678
11.7 手机程序开发.....	634	实例 366 检测系统中安装的 协议.....	679
实例 344 利用短信猫发送短信.....	634	实例 367 域名解析.....	680
实例 345 利用短信远程关闭 计算机.....	635		
实例 346 使用“猫”拨打电话.....	637		
11.8 其他程序.....	638		
实例 347 利用神龙卡制作练歌房 程序.....	638		
实例 348 指纹识别.....	639		
实例 349 游戏杆控制.....	643		

第 12 章 网络开发技术..... 647

12.1 获取计算机信息.....	648	13.1 上网控制.....	684
实例 350 获取计算机名称和 工作组.....	648	实例 368 定时登录 Internet.....	684
实例 351 通过计算机名获取 IP 地址.....	649	实例 369 根据网络连接控制 IE 启动.....	685
实例 352 获取本机 MAC 地址.....	650	13.2 文件上传与下载.....	686
实例 353 获得系统打开的端口和 状态.....	652	实例 370 FTP 文件上传程序.....	686
		实例 371 HTTP 服务器多线程 文件下载.....	689
		实例 372 遍历 FTP 文件目录.....	690
		13.3 邮件管理.....	692
		实例 373 邮件接收程序.....	692
		实例 374 发送电子邮件附件.....	693
		实例 375 使用 MAPI 发送邮件.....	695
		13.4 上网监控.....	697
		实例 376 监控上网过程.....	697
		实例 377 网络监听工具.....	698

第 13 章 Web 编程..... 683

13.5 浏览器应用 702

实例 378 制作自己的网络浏览
软件 702实例 379 XML 数据库文档的
浏览 704实例 380 使用 WebBrowser 执行
脚本 706

实例 381 电子书阅读器 707

13.6 网上信息提取 713

实例 382 定时提取网页源码 714

实例 383 网上天气预报 715

实例 384 网页链接提取器 716

13.7 其他 718

实例 385 利用 TAPI 实现网络
拨号 718

实例 386 互联网文件传输 720

第 14 章 加密、安全与软件注册 729

14.1 数据加密与解密 730

实例 387 数据加密技术 730

实例 388 使用 MD5 算法对
密码进行加密 731实例 389 对数据报进行加密保障
通信安全 735实例 390 对档案进行加密和
解密 737

14.2 软件注册与加密 742

实例 391 利用 INI 文件对软件
进行注册 742实例 392 利用注册表设计软件
注册程序 744实例 393 利用网卡序列号设计
软件注册程序 745实例 394 根据 CPU 和磁盘序列号
设计软件注册程序 747

第 15 章 实用工具 749

实例 395 实现纪念日提醒 750

实例 396 SQL 数据库提取器 751

实例 397 加班网上管理 758

实例 398 垃圾文件清理工具 761

实例 399 网页照相机 763

实例 400 屏幕截图工具 766

附录 技术要点对应实例位置 771

第 1 章

窗体与界面设计



- 菜单应用实例
- 弹出菜单应用实例
- 工具栏应用实例
- 状态栏应用实例
- 导航界面应用实例
- 界面窗体应用实例
- 多媒体宣传光盘应用实例
- 多媒体触摸屏程序应用实例
- 窗体位置应用实例
- 窗体标题栏应用实例
- 窗体形状及应用
- 通用对话框的应用

Visual C++

1.1 菜单应用实例

在设计程序主界面时,菜单几乎是必不可少的界面元素之一。在 MFC 中,提供了 CMenu 类用于操作和管理菜单。本节将通过几个典型实例介绍各种常用菜单的设计。

实例 001 在系统菜单中添加菜单项

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\01\001

实例说明

系统菜单是用户右击标题栏时弹出的快捷菜单。默认情况下,系统菜单只包含少数与标题栏按钮对应的菜单项,如何在系统菜单中添加自己的菜单项呢?本实例实现了该功能,效果如图 1.1 所示。

技术要点

为了操作系统菜单,首先需要获取一个系统菜单指针,这可以通过 GetSystemMenu 函数实现,然后利用菜单指针添加一个菜单项,最后在对话框的 OnSysCommand 方法中处理菜单项的命令。

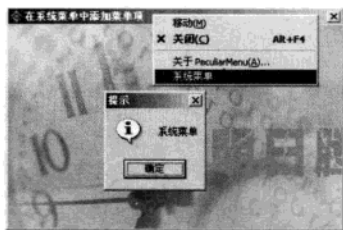


图 1.1 在系统菜单中添加菜单项

GetSystemMenu 方法用于获取一个系统菜单指针,语法如下:

```
CMenu* GetSystemMenu( BOOL bRevert ) const;
```

参数说明:

- bRevert: 确定方法执行的动作,如果为 FALSE,该方法返回当前正在使用的系统菜单,如果为 TRUE,该方法将重新设置系统菜单到默认状态,并且方法返回值不可用。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框类中定义一个菜单指针 m_pMenu,用于指向系统菜单。
- (3) 主要程序代码。

在对话框初始化时 (OnInitDialog 函数中) 获取系统菜单指针,向系统菜单中添加菜单项,代码如下:

```
m_pMenu = GetSystemMenu(FALSE); //获得正在使用的系统菜单指针  
m_pMenu->AppendMenu(MF_STRING,IDI_PECULIARMENU,"系统菜单"); //添加菜单项
```

响应菜单项的命令消息,在对话框的 OnSysCommand 方法中添加消息处理代码:

```
void CPeculiarMenuDlg::OnSysCommand(UINT nID, LPARAM lParam)  
{  
    if ((nID) == IDM_ABOUTBOX)  
    {  
        CAboutDlg dlgAbout;  
        dlgAbout.DoModal();  
    }  
    else if (nID == IDI_PECULIARMENU) //如果是添加的菜单项  
    {  
        MessageBox("系统菜单","提示",MB_OK|MB_ICONINFORMATION); //弹出消息提示  
    }  
    else  
    {  
        CDialog::OnSysCommand(nID, lParam);  
    }  
}
```

举一反三

根据本实例，读者可以：

- 禁用系统菜单项。

实例 002 带图标的程序菜单

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\01\002

实例说明

在 MFC 应用程序中，默认情况下，CMenu 类并不具有显示图标的功能，但在许多应用程序中，菜单中都带有漂亮的图标。那么如何在 MFC 应用程序中为菜单添加图标呢？本实例实现了该功能，效果如图 1.2 所示。

技术要点

要实现带图标的菜单，需要从 CMenu 类派生一个子类，并在子类中改写 DrawItem 方法和 MeasureItem 方法。基本设计思路如下。

首先定义一个记录菜单项信息的结构 CMenuItemInfo，该结构包含了菜单项的文本、图像索引、ID 等信息。然后从 CMenu 中派生一个子类，本实例为 CIconMenu。在该类中定义一个方法 ChangeMenuItem，利用递归的方式修改所有的菜单项信息，使其具有自绘风格 (MF_OWNERDRAW)。接着在 CIconMenu 类中定义绘制菜单项文本、绘制菜单项图标和绘制分隔条的方法。最后改写 MeasureItem 方法，设置菜单项的大小，改写 DrawItem 方法，根据菜单项的不同状态，绘制菜单项。

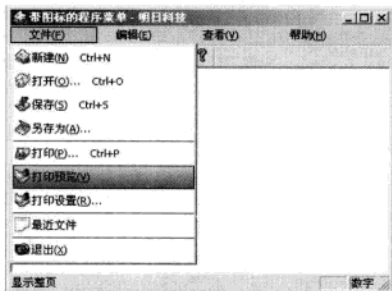


图 1.2 带图标的程序菜单

实现过程

- (1) 新建一个单文档/视图结构的应用程序。
- (2) 从 CMenu 类派生一个子类 CIconMenu。
- (3) 定义一个菜单项结构 CMenuItemInfo，代码如下：

```
/******  
CMenuItemInfo结构用于记录菜单项信息  
******/  
struct CMenuItemInfo  
{  
    CString m_ItemText;    //菜单项文本  
    int m_IconIndex;    //菜单项索引  
    int m_ItemID;    //菜单标记 -2顶层菜单， -1弹出式菜单， 0分隔条，其他普通菜单  
};
```

(4) 在 CIconMenu 类中定义一个 CImageList 类型的成员变量 m_imagelist，用于存储图像；定义一个 CMenuItemInfo 结构数组 m_ItemLists，用于记录每个菜单项信息。

- (5) 在 CIconMenu 类的构造函数中创建图像列表，并向图像列表中添加图标，代码如下：

```
CIconMenu::CIconMenu()  
{  
    m_index= 0;  
    m_IconIndex= 0;  
    //创建图像列表  
    m_imagelist.Create(16,16,ILC_COLOR24|ILC_MASK,0,0);  
    //添加图标  
    m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON1));  
    m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON2));  
    m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON3));  
    m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON4));  
}
```

```
m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON5));
m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON6));
m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON7));
m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON8));
m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON9));
m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON10));
}
```

(6) 向 CIconMenu 类中添加 ChangeMenuItem 方法, 修改菜单项信息, 代码如下:

```
BOOL CIconMenu::ChangeMenuItem(CMenu* m_menu, BOOL m_Toped)
{
    if (m_menu != NULL)
    {
        int m_itemcount = m_menu->GetMenuItemCount(); //获得菜单项目数
        for (int i=0; i<m_itemcount; i++)
        {
            m_menu->GetMenuString(i, m_ItemLists[m_index].m_ItemText, MF_BYPOSITION); //获得菜单字符串
            int m_itemID = m_menu->GetMenuItemID(i); //获得菜单ID
            if (m_itemID == -1 && m_Toped)
            {
                m_itemID = -2; //顶层菜单
            };
            m_ItemLists[m_index].m_ItemID = m_itemID; //保存菜单ID
            if (m_itemID > 0)
            {
                m_ItemLists[m_index].m_IconIndex = m_iconindex; //保存图标索引
                m_iconindex += 1; //设置索引增加
            }
            m_menu->ModifyMenu(i, MF_OWNERDRAW|MF_BYPOSITION|MF_STRING,
                m_ItemLists[m_index].m_ItemID, (LPCTSTR)&(m_ItemLists[m_index])); //修改菜单风格
            m_index += 1;
            CMenu* m_subMenu = m_menu->GetSubMenu(i); //获得子菜单
            if (m_subMenu)
            {
                ChangeMenuItem(m_subMenu); //递归修改菜单项信息
            }
        }
    }
    return TRUE;
}
```

(7) 向 CIconMenu 类中添加 DrawComMenu 方法, 绘制菜单项, 代码如下:

```
void CIconMenu::DrawComMenu(CDC* m_pdc, CRect m_rect, COLORREF m_fromcolor, COLORREF
m_tocolor, BOOL m_selected)
{
    if (m_selected) //选中
    {
        m_pdc->Rectangle(m_rect); //绘制矩形区域
        m_rect.DeflateRect(1, 1); //缩小区域
        int r1, g1, b1;
        //读取渐变起点的颜色值
        r1 = GetRValue(m_fromcolor);
        g1 = GetGValue(m_fromcolor);
        b1 = GetBValue(m_fromcolor);
        int r2, g2, b2;
        //读取渐变终点的颜色值
        r2 = GetRValue(m_tocolor);
        g2 = GetGValue(m_tocolor);
        b2 = GetBValue(m_tocolor);
        float r3, g3, b3; //菜单区域水平方向每个点RGB值应该变化的度 (范围)
        r3 = ((float)(r2-r1)) / (float)(m_rect.Height());
        g3 = ((float)(g2-g1)) / (float)(m_rect.Height());
        b3 = ((float)(b2-b1)) / (float)(m_rect.Height());
        COLORREF r, g, b; //菜单区域水平方向每个点的颜色值
        CPen* m_oldpen;
        for (int i = m_rect.top; i < m_rect.bottom; i++) //绘制渐变色效果
        {
            r = r1 + (int)r3*(i-m_rect.top);
            g = g1 + (int)g3*(i-m_rect.top);
            b = b1 + (int)b3*(i-m_rect.top);
            CPen m_pen (PS_SOLID, 1, RGB(r, g, b)); //创建画笔
            m_oldpen = m_pdc->SelectObject(&m_pen); //选择画笔
            m_pdc->MoveTo(m_rect.left, i); //设置画线起点
            m_pdc->LineTo(m_rect.right, i); //画线
        }
        m_pdc->SelectObject(m_oldpen);
    }
    else
    {

```

```
{
    m_pdc->FillSolidRect(m_rect,RGB(0x000000F9,0x000000F8,0x000000F7)); //填充区域
}
}
```

(8) 向 CIconMenu 类中添加 DrawMenuItem 方法, 绘制菜单项图标, 代码如下:

```
void CIconMenu::DrawMenuItem(CDC* m_pdc,CRect m_rect,int m_icon)
{
    m_imagelist.Draw(m_pdc,m_icon,CPoint(m_rect.left+2,m_rect.top+4),ILD_TRANSPARENT); //绘制图标
}
}
```

(9) 改写 MeasureItem 虚拟方法, 设置菜单项大小, 代码如下:

```
void CIconMenu::MeasureItem(LPMEASUREITEMSTRUCT lpStruct)
{
    if (lpStruct->CtlType==ODT_MENU)
    {
        lpStruct->itemHeight = ITEMHEIGHT; //设置菜单项高度
        lpStruct->itemWidth = ITEMWIDTH; //设置菜单项宽度
        CMenultemInfo* m_iteminfo;
        m_iteminfo = (CMenultemInfo*)lpStruct->itemData; //获得菜单项数据
        lpStruct->itemWidth = ((CMenultemInfo*)lpStruct->itemData)->m_ItemText.GetLength()*10; //设置宽度
        switch(m_iteminfo->m_ItemID)
        {
            case 0: //分隔条
            {
                lpStruct->itemHeight = 1; //设置高度为1
                break;
            }
        }
    }
}
}
```

(10) 改写 DrawItem 虚拟方法, 绘制菜单项, 代码如下:

```
void CIconMenu::DrawItem(LPDRAWITEMSTRUCT lpStruct)
{
    if (lpStruct->CtlType==ODT_MENU)
    {
        if(lpStruct->itemData == NULL) return;
        unsigned int m_state = lpStruct->itemState; //获得菜单状态
        CDC* m_dc = CDC::FromHandle(lpStruct->hDC); //获得设备上下文

        CString str = ((CMenultemInfo*)(lpStruct->itemData))->m_ItemText; //获得菜单项文本
        LPSTR m_str = str.GetBuffer(str.GetLength());

        int m_itemID = ((CMenultemInfo*)(lpStruct->itemData))->m_ItemID; //获得菜单项ID
        int m_itemicon = ((CMenultemInfo*)(lpStruct->itemData))->m_IconIndex; //获得图标索引
        CRect m_rect = lpStruct->rcItem; //获得菜单项区域

        m_dc->SetBkMode(TRANSPARENT); //设置背景透明

        switch(m_itemID)
        {
            case -2: //顶层菜单
            {
                DrawTopMenu(m_dc,m_rect,(m_state&ODS_SELECTED)|(m_state&0x0040)); //0x0040 == ODS_HOTLIGHT
                DrawItemText(m_dc,m_str,m_rect); //绘制文本
                break;
            }
            case -1: //弹出式菜单
            {
                DrawItemText(m_dc,m_str,m_rect); //绘制文本
                break;
            }
            case 0: //分隔条
            {
                DrawSeparator(m_dc,m_rect); //绘制分隔条
                break;
            }
            default: //普通菜单
            {
                DrawComMenu(m_dc,m_rect,0xf00f,m_state&ODS_SELECTED); //绘制菜单背景
                DrawItemText(m_dc,m_str,m_rect); //绘制菜单文本
                DrawMenuItem(m_dc,m_rect,m_itemicon); //绘制菜单项图标
                break;
            }
        }
    }
}
}
```


举一反三

根据本实例，读者可以：

- 实现具有状态栏提示功能的菜单。

实例 003 根据 INI 文件创建菜单

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\01\003

实例说明

为了增强应用程序的灵活性，菜单有时需要动态创建，创建菜单所需的数据可以有多种存储方式，本实例实现使用 ini 文件来存储菜单数据。以后需要增加或减少菜单项时，只需修改 INI 文件即可。效果如图 1.3 所示。

技术要点

INI 文件中的数据是按节进行存储的，每节都有一个节名，节下可以有若干个键，每个键都对应一个数据值。如图 1.4 所示，用 INI 文件结构来保存菜单数据。



图 1.3 根据 INI 文件创建菜单

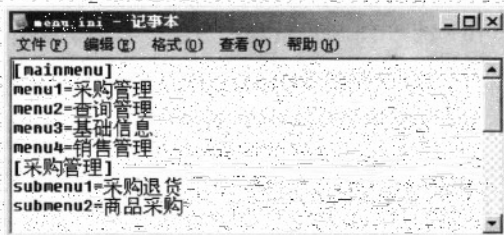


图 1.4 菜单的 INI 文件

INI 文件中每节下的键的数量是不固定的，要使用 GetPrivateProfileSection 函数对节下的键以及键对应的数据进行枚举。

GetPrivateProfileSection 函数是获取节下的所有数据，语法如下：

```
DWORD GetPrivateProfileSection(LPCTSTR lpAppName,  
LPTSTR lpReturnedString, DWORD nSize, LPCTSTR lpFileName);
```

参数说明：

- lpAppName：要获取的节名。
- lpReturnedString：存放返回值字符串指针，返回值有可能是多个结构，结构和结构之间是回车换行符。
- nSize：设置将要保存的字符串的大小。
- lpFileName：INI 文件名，可以是全路径，如果不是全路径，默认就在系统文件夹下新建一个 INI 文件。

GetPrivateProfileSectionNames：该函数是从 ini 文件中获取所有节的名称。语法：

```
DWORD GetPrivateProfileSectionNames(  
LPTSTR lpszReturnBuffer, DWORD nSize, LPCTSTR lpFileName);
```

参数说明：

- lpszReturnBuffer：存放返回值的字符串指针。
- nSize：设置将要保存的字符串的大小。
- lpFileName：INI 文件名，可以是全路径，如果不是全路径默认就在系统文件夹下新建

一个 INI 文件。

实现过程

- (1) 新建基于对话框的应用程序。
- (2) 在工程中添加 LoadSubMenu、IsHaveSubMenu 和 CreateMenuFromFile 3 个自定义函数，分别实现加载子菜单、判断是否有子菜单和根据 INI 文件创建菜单。
- (3) 自定义函数 CreateMenuFromFile 负责读取 INI 文件并创建菜单，函数实现代码如下：

```
void CCreateIniMenuDlg::CreateMenuFromFile()
{
    CString strFilePath=".\\menu.ini";
    CString strSectionName="mainmenu";
    _TCHAR buf[10240];
    DWORD readlen=::GetPrivateProfileSection(strSectionName,buf,12040,strFilePath);//列举INI文件中所包含的节
    _TCHAR *pbuf=buf;
    size_t size=strlen(pbuf);
    while(size)
    {
        CString strTmp(pbuf);
        CString strRight;
        int iRightPos=strTmp.Find("=");           //在字符串中查找“=”符号
        strRight=strTmp.Mid(iRightPos+1);         //获取字符串中“=”符号右面的字符串
        LoadSubMenu(&m_cMenu,strRight);          //加载子菜单
        pbuf+=size+1;
        size=strlen(pbuf);
    }
    SetMenu(&m_cMenu);                           //设置对话框菜单
}
```

举一反三

根据本实例，读者可以：

- 实现动态菜单响应。

实例 004 浮动的菜单

这是一个可以提高分析能力的实例

实例位置：光盘\mingrisoft\01\004

实例说明

在许多应用软件中，菜单都具有浮动功能。例如在 Word 中，用户可以将菜单拖动到任意位置。本实例设计了一个浮动的菜单，效果如图 1.5 所示。

技术要点

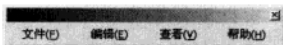


图 1.5 浮动的菜单

在文档/视图结构的应用程序中，默认情况下，工具栏具有拖动菜单的功能。在用户单击工具栏按钮时，能够弹出一个快捷菜单，就可以实现一个浮动的“菜单”了，如何确定用户是否单击了工具栏按钮呢，可以通过在框架窗口中添加 TBN_DROPDOWN 通知消息映射宏来实现。

```
ON_NOTIFY(TBN_DROPDOWN,AFX_IDW_TOOLBAR,OnToolbarDropDown)
```

这样，当用户单击工具栏按钮时，就会调用自定义的 OnToolbarDropDown 方法，在该方法中可以获得用户单击的工具栏按钮 ID 和按钮的客户区域，根据按钮 ID 加载相应的子菜单，根据按钮区域设置子菜单弹出的位置。

实现过程

- (1) 新建一个文档/视图应用程序。
- (2) 从 CToolBar 类派生一个子类 CFloatMenu。在 CFloatMenu 中定义一个 CMenu 指针 m_pSubMenu。

(3) 向 CFloatMenu 类中添加 AddButtonFromMenu 方法, 该方法根据菜单添加工具栏按钮, 代码如下:

```
BOOL CFloatMenu::AddButtonFromMenu(UINT IDresource)
{
    CMenu Menu;
    if (Menu.LoadMenu(IDresource))
    {
        //获取菜单顶层菜单数
        UINT ButtonCount = Menu.GetMenuItemCount();
        UINT * array = new UINT[ButtonCount];
        CString text;
        for (int n = 0; n < ButtonCount; n++)
        {
            array[n] = ID_BUTTON1 + n;           //设置工具栏按钮ID
        }
        //添加工具栏按钮
        SetButtons(array, ButtonCount);
        for (int i = 0; i < ButtonCount; i++)
        {
            Menu.GetMenuString(i, text, MF_BYPOSITION); //获得菜单项文本
            SetButtonText(i, text);                     //设置工具栏按钮文本
            SetButtonStyle(i, TBSTYLE_DROPDOWN);        //设置工具栏按钮风格
        }
        delete array;
        Menu.DestroyMenu();
        return true;
    }
    else
        return FALSE;
}
```

(4) 向 CFloatMenu 类中添加 GetIndexFromPoint 方法, 根据光标点确定工具栏按钮索引, 代码如下:

```
UINT CFloatMenu::GetIndexFromPoint(CPoint pot)
{
    CRect rect;
    for (int i = 0; i < GetToolBarCtrl().GetButtonCount(); i++) //根据工具栏按钮数循环
    {
        GetItemRect(i, rect); //获得工具栏按钮区域
        if (rect.PtInRect(pot)) //判断光标是否在工具栏按钮上
            return i; //返回工具栏按钮索引
    }
    return -1;
}
```

(5) 在框架类中添加通知消息映射宏, 代码如下:

```
ON_NOTIFY(TBN_DROPDOWN, AFX_IDW_TOOLBAR, OnToolBarDropDown)
```

(6) 在框架类中添加 OnToolBarDropDown 方法, 在用户单击工具栏按钮时弹出菜单, 代码如下:

```
void CMainFrame::OnToolBarDropDown(NMTOOLBAR *pnmth, LRESULT *plr)
{
    CWnd* pWnd = &m_wndFloatTool; //获得浮动工具栏窗口指针
    UINT nID = IDR_MAINFRAME;
    CMenu menu;
    menu.LoadMenu(nID); //加载菜单资源
    CMenu* pPop = menu.GetSubMenu(pnmth->iItem-ID_BUTTON1); //获得子菜单
    m_wndFloatTool.m_pSubMenu = pPop;
    m_wndFloatTool.MenuPopIndex = pnmth->iItem-ID_BUTTON1; //设置菜单索引
    CRect rc;
    m_wndFloatTool.GetToolBarCtrl().GetItemRect(pnmth->iItem-ID_BUTTON1, rc); //获得工具栏按钮区域
    pWnd->ClientToScreen(&rc); //转换为屏幕坐标区域
    pPop->TrackPopupMenu(TPM_LEFTALIGN|TPM_LEFTBUTTON|TPM_VERTICAL, //显示弹出菜单
        rc.left, rc.bottom, this, &rc);
    m_wndFloatTool.MenuPopIndex = -1;
}
```

举一反三

根据本实例, 读者可以:

- 设计浮动的对话框窗口。

1.2 弹出菜单应用实例

在程序中使用弹出式菜单能够方便用户操作。本节通过几个典型实例介绍各种弹出式菜单的设计。

实例 005

在控件上单击右键弹出菜单

本实例可以方便操作、提高效率

实例位置：光盘\mingrisoft\01\005

实例说明

在许多应用软件中，当用户单击鼠标右键时，会弹出一个快捷菜单，用户可以通过快捷菜单方便地进行各种操作。本实例实现了弹出式菜单的功能，效果如图 1.6 所示。

技术要点

实现弹出式菜单非常简单，只需要处理 WM_CONTEXTMENU 消息就可以了。在其消息处理函数（默认为 OnContextMenu）中调用菜单的 TrackPopupMenu 方法即可在指定位置弹出菜单。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加 List Control 控件，设置控件属性，如图 1.7 所示。

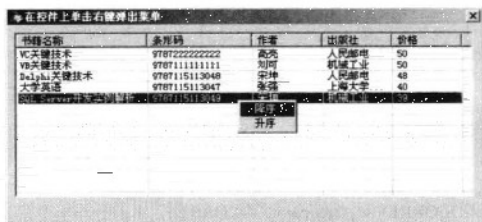


图 1.6 在控件上单击右键弹出菜单

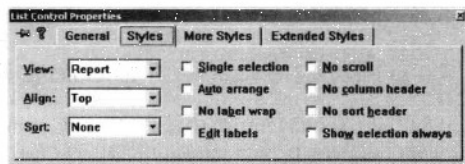


图 1.7 列表控件属性

- (3) 处理对话框的 WM_CONTEXTMENU 消息，代码如下：

```
void CPopManuDlg::OnContextMenu(CWnd* pWnd, CPoint point)
{
    CMenu m_popmenu;                                //定义菜单对象
    m_popmenu.LoadMenu(IDR_POPMENU);                 //加载菜单资源
    CMenu* m_submenu = m_popmenu.GetSubMenu(0);       //获得子菜单
    m_submenu->TrackPopupMenu(TPM_LEFTBUTTON|TPM_LEFTALIGN, point.x, point.y, this); //显示弹出菜单
    m_popmenu.DestroyMenu();
}
```

举一反三

根据本实例，读者可以：

- 设计系统托盘菜单。

实例 006

个性化的弹出菜单

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\01\006

实例说明

网上许多软件的弹出式菜单非常漂亮。本实例设计一个漂亮的弹出式菜单，效果如图 1.8 所示。

技术要点

设计弹出式菜单与设计普通的菜单一样, 需要从 CMenu 类派生一个子类, 然后改写 MeasureItem 方法设置菜单项大小; 改写 DrawItem 方法根据当前状态绘制菜单。设计思路可以参考实例 002 带图标的程序菜单。

实现过程

(1) 新建一个基于对话框的应用程序。

(2) CMenu 派生一个子类 CiconMenu, 定义一个菜单项结构 CMenuItemInfo, 代码如下:

```

/*****
CMenuItemInfo结构用于记录菜单项信息
*****/
struct CMenuItemInfo
{
    CString m_ItemText;           //菜单项文本
    int m_IconIndex;             //菜单项索引
    int m_ItemID;                //菜单标记 -2顶层菜单, -1弹出式菜单, 0分隔条, 其他普通菜单
};

```

(3) 在 CIconMenu 类中定义如下成员变量:

```

CMenuItemInfo m_ItemLists[MAX_MENU_COUNT]; //菜单项信息
int m_index;                               //临时索引
int m_IconIndex;                           //图像索引
BOOL m_IsDrawTitle;                        //是否重绘标题
CFont m_TitleFont;                         //标题字体

```

(4) 向 CIconMenu 类中添加 DrawItemText 方法, 绘制菜单项文本, 代码如下:

```

/*****
功能描述: 绘制菜单项文本
*****/
参数说明: m_pdc标识画布对象, str标识菜单文本, m_rect标识菜单区域
*****/
void CIconMenu::DrawItemText(CDC* m_pdc, LPCTSTR str, CRect m_rect)
{
    m_rect.DeflateRect(40, 0, 0);           //调整文本绘制区域
    m_pdc->DrawText(str, m_rect, DT_SINGLELINE|DT_VCENTER|DT_LEFT); //绘制菜单项文本
}

```

(5) 向 CIconMenu 类中添加 DrawComMenu 方法, 绘制普通的菜单, 代码如下:

```

void CIconMenu::DrawComMenu(CDC* m_pdc, CRect m_rect, COLORREF m_fromcolor, COLORREF m_tocolor, BOOL m_selected)
{
    if (m_selected) //如果选中菜单项
    {
        m_pdc->SelectStockObject(BLACK_PEN); //设置黑色画笔
        m_rect.DeflateRect(25, 1, 0, 2); //调整菜单项区域
        m_pdc->Rectangle(m_rect); //绘制矩形
        CBitmap m_bitmap; //加载位图资源
        m_bitmap.LoadBitmap(IDB_LEFTBITMAP);
        BITMAP m_size; //获得位图数据
        m_bitmap.GetBitmap(&m_size);
        CDC m_memdc; //创建内存兼容设备上下文
        m_memdc.CreateCompatibleDC(m_pdc);
        CGdiObject* m_oldobject; //选入位图
        m_oldobject = m_memdc.SelectObject(&m_bitmap);
        m_pdc->StretchBlt(m_rect.left+1, m_rect.top+1, m_rect.Width()-2, m_rect.Height()-2, m_memdc, 0, 0, m_size.bmWidth, m_size.bmHeight, SRCCOPY); //绘制位图
        m_bitmap.DeleteObject();
    }
    else
    {
        m_pdc->FillSolidRect(m_rect, RGB(0x000000F9, 0x000000F8, 0x000000F7)); //填充菜单项背景
    }
}

```

(6) 向 CIconMenu 类中添加 DrawSeparator 方法, 绘制分隔条, 代码如下:

```

void CIconMenu::DrawSeparator(CDC* m_pdc, CRect m_rect)
{
    if (m_pdc != NULL)
    {
        m_rect.DeflateRect(25, 0, 0, 0); //设置绘制区域
        m_pdc->Draw3dRect(m_rect, RGB(255, 0, 0), RGB(0, 0, 255)); //绘制滚动条
    }
}

```

商品信息管理
供应商品信息管理
客户信息管理
商品销售管理
销售退货管理
商品销售查询
销售退货查询

图 1.8 个性化的弹出菜单

(7) 向 CIconMenu 类中添加 ChangeMenuItem 方法, 修改菜单项的风格, 代码如下:

```

BOOL CIconMenu::ChangeMenuItem(CMenu* m_menu, BOOL m_Toped)
{
    if (m_menu != NULL)
    {
        int m_itemcount = m_menu->GetMenuItemCount();           //获得菜单项数量
        for (int i=0; i<m_itemcount; i++)
        {
            m_menu->GetMenuString(i, m_ItemLists[m_index].m_ItemText, MF_BYPOSITION); //获得菜单项文本
            int m_itemID = m_menu->GetMenuItemID(i);             //获得菜单项ID
            if (m_itemID == -1 && m_Toped)
            {
                m_itemID = -2;                                     //顶层菜单
            };
            m_ItemLists[m_index].m_ItemID = m_itemID;           //保存菜单项ID
            if (m_itemID > 0)
            {
                m_ItemLists[m_index].m_IconIndex = m_IconIndex;   //保存菜单项索引
                m_IconIndex++;                                     //设置菜单项索引增加
            }
            m_menu->ModifyMenu(i, MF_OWNERDRAW|MF_BYPOSITION|MF_STRING,
                m_ItemLists[m_index].m_ItemID, (LPCTSTR)&(m_ItemLists[m_index])); //修改菜单风格
            m_index++;
            CMenu* m_submenu = m_menu->GetSubMenu(i);             //获得子菜单
            if (m_submenu)
            {
                ChangeMenuItem(m_submenu);                         //递归修改菜单项
            }
        }
    }
    return TRUE;
}

```

(8) 改写 CMenu 类的 MeasureItem 方法, 设置菜单项大小, 代码如下:

```

void CIconMenu::MeasureItem(LPMEASUREITEMSTRUCT lpStruct)
{
    if (lpStruct->CtlType == ODT_MENU)
    {
        lpStruct->itemHeight = ITEMHEIGHT;                      //设置菜单项高度
        lpStruct->itemWidth = ITEMWIDTH;                         //设置菜单项宽度
        CMenuItemInfo* m_iteminfo;
        m_iteminfo = (CMenuItemInfo*)(lpStruct->itemData);       //获得菜单项数据
        lpStruct->itemWidth = ((CMenuItemInfo*)(lpStruct->itemData))->m_ItemText.GetLength()*10; //设置菜单项高度
        switch (m_iteminfo->m_ItemID)
        {
            case 0:                                               //分隔条
            {
                lpStruct->itemHeight = 1;                         //设置分隔条高度为1
                break;
            }
        }
    }
}

```

(9) 改写 CMenu 类的 DrawItem 方法, 根据菜单状态绘制菜单, 代码如下:

```

void CIconMenu::DrawItem(LPDRAWITEMSTRUCT lpStruct)
{
    if (lpStruct->CtlType == ODT_MENU)
    {
        if (lpStruct->itemData == NULL) return;
        unsigned int m_state = lpStruct->itemState;             //获得菜单项状态
        CDC* m_dc = CDC::FromHandle(lpStruct->hDC);             //获得设备上下文
        CString str = ((CMenuItemInfo*)(lpStruct->itemData))->m_ItemText; //获得菜单项文本
        LPCTSTR m_str = str.GetBuffer(str.GetLength());
        int m_itemID = ((CMenuItemInfo*)(lpStruct->itemData))->m_ItemID; //获得菜单项ID
        int m_itemicon = ((CMenuItemInfo*)(lpStruct->itemData))->m_IconIndex; //获得菜单项索引
        CRect m_rect = lpStruct->rcItem;                         //获得菜单项区域
        m_dc->SetBkMode(TRANSPARENT);                          //设置背景透明
        switch (m_itemID)
        {
            case -2:                                               //顶层菜单
            {
                DrawTopMenu(m_dc, m_rect, (m_state & ODS_SELECTED) | (m_state & 0x0040)); //0x0040 == ODS_HOTLIGHT
                DrawItemText(m_dc, m_str, m_rect);               //绘制菜单项文本
                break;
            }
        }
    }
}

```

```

case -1:                                     //弹出菜单
{
    DrawItemText(m_dc,m_str,m_rect);        //绘制菜单项文本
    break;
}
case 0:                                     //分隔条
{
    DrawSeparator(m_dc,m_rect);             //绘制分隔条
    break;
}
default:                                    //普通菜单
{
    DrawComMenu(m_dc,m_rect,0xf00f,0xf00f,m_state&ODS_SELECTED); //绘制菜单背景
    DrawItemText(m_dc,m_str,m_rect);        //绘制菜单项文本
    DrawMenuTitle(m_dc,m_rect,"明日科技有限公司"); //绘制菜单左侧标题
    break;
}
}
}

```

举一反三

根据本实例，读者可以：

- 开发具有背景颜色的菜单。

实例 007 任务栏托盘弹出菜单

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\01\007

实例说明

在安装完瑞星、金山词霸等软件时，在系统的任务栏中会显示一个托盘图标。用户用鼠标右键单击托盘图标，就会弹出一个快捷菜单。本实例将实现一个任务栏托盘菜单，效果如图 1.9 所示。

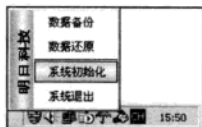


图 1.9 任务栏托盘弹出菜单

技术要点

要设计任务栏托盘菜单，需要使用 Shell_NotifyIcon 函数。该函数语法如下：

```

WINSHLLAPI BOOL WINAPI Shell_NotifyIcon(
    DWORD dwMessage,
    NOTIFYICONDATA pnid
);

```

参数说明：

- dwMessage：表示发送的消息值，其可选值如下。
 - NIM_ADD：表示添加图标到任务栏。
 - NIM_DELETE：表示从任务栏区域删除一个图标。
 - NIM_MODIFY：表示修改任务栏区域的一个图标。
- pnid：是 NOTIFYICONDATA 结构指针。

NOTIFYICONDATA 结构定义如下：

```

typedef struct _NOTIFYICONDATA {
    DWORD cbSize;
    HWND hWnd;
    UINT uID;
    UINT uFlags;
    UINT uCallbackMessage;
    HICON hIcon;
    char szTip[64];
} NOTIFYICONDATA, *PNOTIFYICONDATA;

```

参数说明：

- cbSize 确定 NOTIFYICONDATA 结构的大小。

- hWnd 表示接收任务栏菜单消息的窗口句柄。
- uID 表示任务栏图标标识符。
- uFlags 确定 NOTIFYICONDATA 结构中哪些成员是合法的。
- uCallbackMessage 表示应用程序定义的消息标识符，系统将要发送该消息到 hWnd 表示的窗口。
- hIcon 表示添加、修改或删除的图标句柄。
- szTip 是工具提示文本。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 从 CMenu 类派生一个子类 CIconMenu，改写 DrawItem 方法和 MeasureItem 方法重新绘制菜单。
- (3) 在对话框类中定义一个 NOTIFYICONDATA 结构变量 m_traydata，在 OnInitDialog 方法中初始化 m_traydata。

```
m_traydata.cbSize = sizeof(NOTIFYICONDATA);           //设置结构大小
m_traydata.hIcon = AfxGetApp()->LoadIcon(IDI_TRAYICON); //加载图标资源
m_traydata.hWnd = m_hWnd;                             //设置窗口句柄
char *m_str = "系统管理";                             //声明字符串
//strlen +1 表示将空字符串复制到目标字符串中
strncpy(m_traydata.szTip, m_str, strlen(m_str)+1);    //复制字符串
m_traydata.uCallbackMessage = WM_TRAYMESSAGE;        //设置回传消息
m_traydata.uFlags = NIF_ICON|NIF_MESSAGE|NIF_TIP;     //设置合法参数
```

- (4) 在对话框的 OnSysCommand 方法中判断用户是否单击了“最小化”按钮，如果是，隐藏对话框，调用 Shell_NotifyIcon 函数向任务栏中添加图标，代码如下：

```
void CTrayPopupMenuDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else if ((nID & 0xFFF0) == SC_MINIMIZE)              //如果是最小化消息
    {
        ShowWindow(SW_HIDE);                            //隐藏窗口
        Shell_NotifyIcon(NIM_ADD, &m_traydata);          //添加系统托盘图标
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}
```

- (5) 在对话框的消息映射部分添加映射宏。

ON_MESSAGE(WM_TRAYMESSAGE, OnTrayMessage)

- (6) 向对话框中添加 OnTrayMessage 方法，代码如下：

```
void CTrayPopupMenuDlg::OnTrayMessage(WPARAM wParam, LPARAM lParam)
{
    if (lParam == WM_LBUTTONDOWN)                      //如果在托盘图标上单击鼠标左键
    {
        ShowWindow(SW_RESTORE);                        //恢复显示窗口
    }
    else if (lParam == WM_RBUTTONDOWN)                  //如果在托盘图标上单击鼠标右键
    {
        CPoint m_point;
        ::GetCursorPos(&m_point);                      //获得鼠标当前位置
        CIconMenu* m_submenu = (CIconMenu*)m_menu.GetSubMenu(0); //获得子菜单
        m_submenu->TrackPopupMenu(TPM_LEFTALIGN|TPM_RIGHTBUTTON,
            m_point.x, m_point.y, AfxGetApp()->m_pMainWnd, TPM_LEFTALIGN); //显示弹出菜单
    }
}
```


举一反三

根据本实例,读者可以:

- 修改和删除系统托盘菜单。

1.3 工具栏应用实例

工具栏是应用程序界面的重要组成元素之一,它包含了一组命令按钮,用于执行某些菜单项的功能。通常情况下,将应用程序中常用的功能放置在工具栏中,这样可以方便用户操作,省去了在级联菜单中层层查找菜单项的困扰。在 MFC 类库中,CToolBar 类封装了工具栏的基本功能,在本节中,将详细介绍利用 CToolBar 类设计各种工具栏。

工具栏在开发应用程序时经常用到,本节将介绍几种特色工具栏的实现方法。

实例 008 根据菜单创建工具栏

本实例可以美化界面、简化操作

实例位置: 光盘\mingrisoft\01\008

实例说明

通常工具栏能够实现的功能菜单也能实现,菜单有的命令 ID、图标、名称工具栏也有,如果菜单项不是很多的话可以根据每个菜单项都创建一个工具栏按钮,实例就是实现根据菜单项创建工具栏。效果如图 1.10 所示。

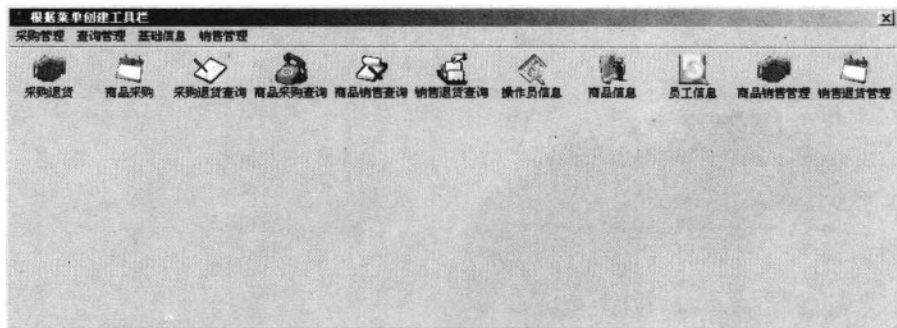


图 1.10 根据菜单创建工具栏

技术要点

实例的关键技术是如何获取所有的菜单项,首先使用 CMenu 类的 LoadMenu 方法加载指定 ID 的菜单资源。然后使用 GetMenuItemCount 获取菜单项的个数,然后使用 GetSubMenu 方法获取子菜单项,如果是级联菜单就继续获取子菜单下的菜单项个数,并遍历子菜单的菜单项,最后通过 GetMenuItemInfo 获取菜单项的内容并生成工具栏按钮。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框初始化函数 OnInitDialog 中根据菜单创建工具栏,代码如下:

```
BOOL CCreateToolBarFromMenuDlg::OnInitDialog()  
{  
    CDialog::OnInitDialog();
```

```
...//代码省略
m_imagelist.Create(32,32,ILC_COLOR32|ILC_MASK,0,0);//创建图像列表
CString strpath;
HICON hicon;
int j;
// 向列表中加载图像
for(j=1;j<10;j++)
{
    strpath.Format(".\\res\\toolbar\\%02d.ico",j);
    hicon = (HICON)::LoadImage(NULL,strpath,IMAGE_ICON,32,32,LR_LOADFROMFILE);
    m_imagelist.Add(hicon);
}
m_toolbar.Create(WS_CHILD|WS_VISIBLE,CRect(0,0,0,0),this,154230);           //创建工具栏
m_toolbar.EnableAutomation();                                              //工具栏支持自动化
m_toolbar.SetImageList(&m_imagelist);                                       //设置工具栏图像列表
TBBUTTON button[11];
int i;
for(i=0;i<11;i++)
{
    button[i].dwData=0;
    button[i].fsState=TBSTATE_ENABLED;                                     //工具栏按钮可用
    button[i].fsStyle=TBSTYLE_BUTTON;                                       //工具栏为按钮样式
}
int iMenuButtonCount=0;
MENUITEMINFO info;
CString strMenuName;
CMenu menDlgMenu;
CMenu *menDlgSubmenu;
menDlgMenu.LoadMenu(IDR_MYMENU);                                           //加载资源中的菜单
int iMenuCount=menDlgMenu.GetMenuItemCount();                             //父菜单数量
for(j=0;j<iMenuCount;j++)
{
    menDlgSubmenu=menDlgMenu.GetSubMenu(j);                                //获取子菜单
    int iSubMenuCount=menDlgSubmenu->GetMenuItemCount();                  //获取子菜单个数
    for(i=0;i<iSubMenuCount;i++)
    {
        menDlgSubmenu->GetMenuString(i,strMenuName,MF_BYPOSITION);        //获取菜单项的名称
        button[iMenuButtonCount].idCommand=menDlgSubmenu->GetMenuItemID(i);
        button[iMenuButtonCount].iBitmap=iMenuButtonCount%9;
        button[iMenuButtonCount].iString=m_toolbar.AddStrings(strMenuName);
        iMenuButtonCount++;
        if(iMenuButtonCount>10)                                           //不能超过TBBUTTON数组的最大值
            break;
    }
    if(iMenuButtonCount>10)
        break;
}
this->SetMenu(&menDlgMenu);                                                 //设置对话框菜单
m_toolbar.AddButtons(iMenuButtonCount,button);                            //添加工具栏按钮
m_toolbar.AutoSize();                                                      //工具栏自动调整大小
m_toolbar.SetStyle(TBSTYLE_FLAT|CCS_TOP);                                  //设置工具栏样式
return TRUE;
```

举一反三

根据本实例，读者可以：

- 分别创建每个菜单下的工具栏。

实例 009 带图标的工具栏

这是一个可以启发思维的实例

实例位置：光盘\mingrisoft\01\009

实例说明

默认情况下，MFC 中提供的工具栏只能显示简单的图像。本实例实现了一个带有图标的工

具栏按钮, 效果如图 1.11 所示。

技术要点

工具栏 CToolBar 提供了一个 GetToolBarCtrl 方法, 用于获得一个 CToolBarCtrl 对象, 该对象提供了一个 SetImageList 方法, 用于设置与工具栏关联的图像列表控件。只要在程序中创建一个图像列表, 并向图像列表中添加图标, 将其与工具栏关联, 那么工具栏按钮就会显示出图像。

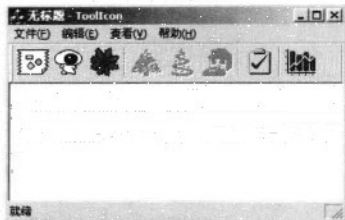


图 1.11 带图标的工具栏

实现过程

- (1) 新建一个文档/视图结构的应用程序。
- (2) 在框架类中定义一个 CImageList 对象 m_ImageList。
- (3) 在框架类的 OnCreate 方法中创建图像列表, 并向图像列表中添加图标。创建工具栏, 将工具栏与图像列表关联, 设置工具栏按钮的大小, 代码如下:

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    //创建图像列表,向图像列表中添加图标
    m_ImageList.Create(32,32,ILC_COLOR24|ILC_MASK,0,1);
    for (int i=0;i<9;i++)
    {
        m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON1+i)); //加载图标资源
    }
    //创建工具栏
    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD|WS_VISIBLE|CBRS_TOP
        |CBRS_GRIPPER|CBRS_TOOLTIPS|CBRS_FLYBY|CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1; // fail to create
    }

    m_wndToolBar.GetToolBarCtrl().SetImageList(&m_ImageList); //管理图像列表
    m_wndToolBar.GetToolBarCtrl().SetButtonSize(CSize(40,40)); //设置按钮大小
    m_wndToolBar.GetToolBarCtrl().SetBitmapSize(CSize(30,30)); //设置图像大小

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1; // fail to create
    }

    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);
    return 0;
}
```

举一反三

根据本实例, 读者可以:

- 实现具有热点效果的工具栏。

实例 010 定制浮动工具栏

本实例可以美化界面、简化操作

实例位置: 光盘\mingrisoft\01\010

实例说明

Microsoft Visual C++中对话框资源控件的窗体就是浮动的工具栏, 在 PhotoShop、Flash 等

软件中也能看到浮动的工具栏窗体。本实例实现的就是一个浮动的工具栏窗体，效果如图 1.12 所示。

技术要点

通过 MFC 向导生成的单文档或多文档应用程序，只要将工具栏向客户区拖动，就可以使工具栏浮动，本实例通过 Create 方法创建一个工具栏，然后通过 FloatControlBar 方法控制工具为浮动的工具栏。



图 1.12 浮动工具栏

CToolBar 类的 Create 方法用于创建一个工具栏，语法如下：

```
BOOL Create(CWnd* pParentWnd, DWORD dwStyle = WS_CHILD | WS_VISIBLE | CBRS_TOP, UINT nID = AFX_IDW_TOOLBAR);
```

参数说明：

- pParentWnd：父窗体指针。
- dwStyle：窗体的样式，默认情况下取值是 WS_CHILD，WS_VISIBLE，CBRS_TOP 3 个，还有如下取值：
 - CBRS_TOP：控制工具栏在顶部。
 - CBRS_BOTTOM：控制工具栏在底部。
 - CBRS_NOALIGN：控制工具栏不对齐。
 - CBRS_TOOLTIPS：工具栏具有提示条。
 - CBRS_SIZE_DYNAMIC：工具栏的大小可以改变。
 - CBRS_SIZE_FIXED：工具栏的大小固定。
 - CBRS_FLOATING：工具栏浮动。
 - CBRS_FLYBY：工具栏平坦样式。
 - CBRS_HIDE_INPLACE：工具栏不显示。
- nID：工具栏在工程中的资源 ID，可以使用默认 ID 值 AFX_IDW_TOOLBAR。。

CFrameWnd 类的 FloatControlBar 方法用于控制工具栏显示的位置及样式，语法如下：

```
CFrameWnd* FloatControlBar(CControlBar* pBar, CPoint point, DWORD dwStyle = CBRS_ALIGN_TOP);
```

参数说明：

- pBar：控制栏指针。
- point：工具栏所在的控制栏左顶点的显示位置。
- dwStyle：工具栏显示样式。

实现过程

- (1) 新建基于单文档视图结构的应用程序。
- (2) 在函数 OnCreate 中将原有的工具栏设置成浮动工具栏，代码如下：

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    DWORD dwStyle = WS_CHILD | WS_VISIBLE;
    dwStyle |= CBRS_FLOATING; //添加浮动样式
    m_wndToolBar.Create(this, dwStyle, AFX_IDW_TOOLBAR); //创建工具栏
    m_wndToolBar.LoadToolBar(IDR_MAINFRAME); //加载工具栏
    if (!m_wndStatusBar.Create(this))
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT))
    {
        TRACE0("Failed to create status bar\n");
        return -1;
    }
    m_wndToolBar.EnableDocking(0); //取消工具栏的停靠方式
    EnableDocking(0); //框架内子窗体也不进行停靠设置
    CRect rect;
```



```
GetWindowRect(&rect);
CPoint point(rect.left+100,rect.top+100);
FloatControlBar(&m_wndToolBar,point,CBRS_ALIGN_LEFT);
return 0;
}
```

```
//获取窗体区域
//计算工具栏显示的坐标
//将工具栏浮动显示
```

举一反三

根据本实例，读者可以：

- 将浮动工具栏固定在窗口的一侧。

实例 011 可调整按钮位置的工具栏

本实例可以美化界面、简化操作

实例位置：光盘\mingrisoft\01\011

实例说明

本实例实现了工具栏上两个按钮互换位置的功能。运行程序，选择菜单“查看”/“改变按钮位置”，程序会将“新建”按钮和“保存”按钮进行位置调换。调换前、后的效果分别如图 1.13 和图 1.14 所示。



图 1.13 调换前

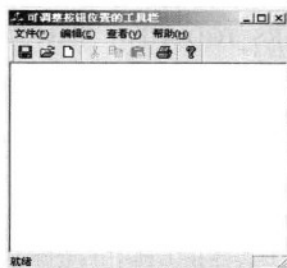


图 1.14 调换后

技术要点

本实例通过 CToolBar 类的 SetButtonInfo 方法实现，SetButtonInfo 方法用来设置工具栏按钮的相关信息，语法如下：

```
void SetButtonInfo( int nIndex, UINT nID, UINT nStyle, int iImage );
```

参数说明：

- nIndex：工具栏上按钮的位置。
- nID：工具栏按钮在工程中的资源 ID 值。
- nStyle：工具栏按钮的风格。
- iImage：工具栏按钮的图片索引值。

实现过程

- (1) 新建名为 ToolbarAjustBtn 的单文档 MFC 工程。
- (2) 修改 Menu 资源 IDR_MAINFRAME，在菜单“查看”下新建子菜单，设置 ID 属性为 ID_VIEW，设置 Caption 属性为“改变按钮位置”。

- (3) 主要程序代码。

菜单 ID_VIEW 的实现函数，实现调用 MoveButton 函数完成工具栏按钮的调整，代码如下：

```
void CMainFrame::OnView()
{
    this->MoveButton(0,2);
}
```

//调用MoveButton函数调整工具栏按钮位置

函数 MoveButton 实现工具栏上不同位置的按钮相互调换,代码如下:

```
void CMainFrame::MoveButton(int oldpos,int newpos)
{
    UINT newID,oldID;
    newID=m_wndToolBar.GetItemID(newpos);           //获得工具栏按钮ID
    oldID=m_wndToolBar.GetItemID(oldpos);           //获得工具栏按钮ID
    m_wndToolBar.SetButtonInfo(oldpos,newID,0,newpos); //设置工具栏按钮信息
    m_wndToolBar.SetButtonInfo(newpos,oldID,0,oldpos); //设置工具栏按钮信息
}
```

举一反三

根据本实例,读者可以:

- 控制工具栏按钮的显示。

实例 012 在工具栏中添加编辑框

本实例可以美化界面、简化操作

实例位置: 光盘\mingrisoft\01\012

实例说明

在工具栏中添加编辑框可以使用户在工具栏上进行输入操作,从而简化用户的操作,可以通过 Create 方法创建编辑框控件,并将编辑框控件的父窗口设为工具栏,在工具栏上显示编辑框。效果如图 1.15 所示。

技术要点

要在工具栏上绘制编辑框,需要在工具栏按钮的位置使用 Create 方法动态创建,Create 是 CEdit 的成员函数,实现编辑框的动态创建,语法如下:

```
BOOL Create( DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID );
```

参数说明:

- dwStyle: 设置编辑框的样式。
- rect: 设置编辑框的显示位置。
- pParentWnd: 设置编辑框的父类。
- nID: 设定一个编辑框的 ID 资源值。

实现过程

- (1) 新建一个基于单文档视图结构的应用程序。
- (2) 在 CMainFrame 类的 OnCreate 函数中创建工具栏,函数实现代码如下:

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    ..... //此处代码省略
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    RECT rect;
    m_Edit.Create(WS_CHILD|WS_CLIPSIBLINGS|WS_EX_TOOLWINDOW|WS_BORDER,
        CRect(0,0,10,10),this,1200);           //创建编辑框
    m_wndToolBar.GetItemRect(11,&rect);           //获取工具栏指定按钮的区域
    m_Edit.SetParent(&m_wndToolBar);           //设置工具栏是编辑框的父窗体
    m_Edit.MoveWindow(&rect);                   //改变便捷框的大小
    m_Edit.ShowWindow(SW_SHOW);                 //显示编辑框
    return 0;
}
```

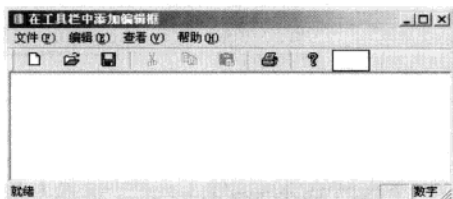


图 1.15 在工具栏中添加编辑框

举一反三

根据本实例,读者可以:

- 在工具栏中添加列表框。

实例 013

动态设置是否显示工具
栏按钮文本

本实例可以美化界面、简化操作

实例位置: 光盘\mingrisoft\01\013

实例说明

工具栏中的按钮并不一定是都显示按钮文本的,用户可以根据当前的操作随时变化,这样可以提高程序的应用性。本实例根据用户的操作来调整工具栏按钮文本的显示。程序首先创建一个新的工具栏,然后根据数据库中的数据来决定哪个工具栏按钮可以显示。工具栏转换前如图 1.16 所示,转换后如图 1.17 所示。



图 1.16 不显示工具栏按钮文本

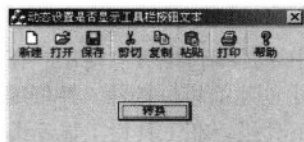


图 1.17 显示工具栏按钮文本

技术要点

本实例中涉及创建工具栏、设置工具栏高度等技术。在程序中使用 CToolBar 来定义一个工具栏对象,通过 GetToolBarCtrl 方法来获取 CToolBarCtrl 类对象。通过 CToolBarCtrl 类对象可以关联图像列表。要设置工具栏按钮高度可以通过 SetHeight 方法来实现,语法如下:

```
void SetHeight( int cyHeight );
```

参数说明:

- cyHeight: 要设置的工具栏高度。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在工程导入 8 个图标资源,并向对话框中添加一个按钮控件。
- (3) 在对话框的头文件中声明变量,代码如下:

```
CToolBar      m_ToolBar;           //工具栏对象
CImageList    m_ImageList;        //列表视图对象
BOOL          m_bText;            //是否显示按钮文本
```

- (4) 在对话框初始化时创建工具栏,代码如下:

```
BOOL CToolTipDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    //...系统代码省略
    //创建图像列表
    m_ImageList.Create(16,16,ILC_COLOR24|ILC_MASK,1,1);
    //向图像列表中添加图标
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON1));
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON2));
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON3));
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON4));
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON5));
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON6));
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON7));
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON8));
}
```

```
UINT array[11];
for(int i=0;i<11;i++)
{
    if(i==3 || i==7 || i==9)
        array[i] = ID_SEPARATOR; //第4、8、10个按钮为分隔条
    else
        array[i] = i+1001;
}
CString str[]={"新建","打开","保存","", "剪切","复制","粘贴","", "打印","", "帮助"};
//创建工具栏
m_ToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
    | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_SIZE_DYNAMIC | CBRS_BORDER_TOP);
m_ToolBar.SetButtons(array,11); //设置工具栏按钮
for(i=0;i<11;i++)
{
    m_ToolBar.SetButtonText(i,str[i]); //设置工具栏按钮文本
}
m_ToolBar.GetToolBarCtrl().SetImageList(&m_ImageList); //关联图像列表
m_ToolBar.SetSizes(CSize(24,24),CSize(16,16)); //设置按钮和图标的大小
RepositionBars(AFX_IDW_CONTROLBAR_FIRST,AFX_IDW_CONTROLBAR_LAST,0);
m_bText = FALSE;
return TRUE;
}
```

(5) 添加 UpdateToolBar 函数，该函数用于设置是否显示工具栏按钮文本，代码如下：

```
void CToolTipDlg::UpdateToolBar(BOOL bUpdate)
{
    if(bUpdate)
    {
        m_ToolBar.SetSizes(CSize(32,32),CSize(16,16)); //设置按钮和图标的大小
        m_ToolBar.SetHeight(36); //设置工具栏高度
        RepositionBars(AFX_IDW_CONTROLBAR_FIRST,AFX_IDW_CONTROLBAR_LAST,0);
    }
    else
    {
        m_ToolBar.SetSizes(CSize(24,24),CSize(16,16)); //设置按钮和图标的大小
        m_ToolBar.SetHeight(28); //设置工具栏高度
        RepositionBars(AFX_IDW_CONTROLBAR_FIRST,AFX_IDW_CONTROLBAR_LAST,0);
    }
}
```

(6) 处理“转换”按钮的单击事件，控制工具栏按钮是否显示，代码如下：

```
void CToolTipDlg::OnButtonupdate()
{
    m_bText = !m_bText;
    UpdateToolBar(m_bText);
}
```

举一反三

根据本实例，读者可以：

- 根据需要设置工具栏按钮是否可用。

实例 014 具有提示功能的工具栏

这是一个可以提高分析能力的实例

实例位置：光盘\mingrisoft\01\014

实例说明

在文档/视图结构的应用程序中，默认情况下，当鼠标在工具栏按钮上停留时，会出现一个工具提示条。运行本实例，将鼠标停留在工具栏的某一个按钮上，即可看到该工具按钮的提示信息，效果如图 1.18 所示。

技术要点

使工具栏具有提示功能，需要同时具备两个条件。



图 1.18 具有提示功能的工具栏

一是工具栏具有 CBRs_TOOLTIPS 风格,二是工具栏的父窗口需要处理 TTN_NEEDTEXT 通知消息。在 MFC 类库中, CFrameWnd 默认处理了 TTN_NEEDTEXT 通知消息,因此,在文档/视图结构的应用程序中,只要工具栏具有 CBRs_TOOLTIPS 风格,就能够显示提示信息。

如果在对话框中添加 TTN_NEEDTEXT 通知消息,需要在消息映射部分添加如下代码:

```
ON_NOTIFY_EX(TTN_NEEDTEXT, 0, OnToolTipNotify)
```

其中, OnToolTipNotify 是处理 TTN_NEEDTEXT 消息的函数,函数原型如下:

```
OnToolTipNotify(UINT id, NMHDR *pNMHDR, LRESULT *pResult);
```

参数说明:

- id: 是发送消息的控件 ID,但此处没有用,因为控件 ID 可以来自于 pNMHDR。
- pNMHDR: 是一个 NMHDR 结构指针(实际应该是 NMTTDISPINFO 结构指针),NMHDR 结构记录了发送消息的控件 ID、句柄等信息。
- pResult: 表示结果代码指针,TTN_NEEDTEXT 消息可以忽略该参数。

实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框类中定义一个 CToolBar 变量 m_wndToolBar。在工作区的资源视图中创建一个工具栏资源,如图 1.19 所示。



图 1.19 工具栏资源设计

(3) 在对话框的 OnInitDialog 方法中创建工具栏,代码如下:

```
//创建工具栏
if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRs_TOP
| CBRs_GRIPPER | CBRs_TOOLTIPS | CBRs_SIZE_DYNAMIC | CBRs_BORDER_TOP) ||
!m_wndToolBar.LoadToolBar(IDR_TOOLBAR1))
{
    TRACE0("Failed to create toolbar\n");
    return -1; // fail to create
}

//设置图像和按钮的大小,以适合演示按钮文本
m_wndToolBar.GetToolBarCtrl().SetBitmapSize(CSize(16,16)); //设置显示图像大小
m_wndToolBar.GetToolBarCtrl().SetButtonSize(CSize(32,32)); //设置工具栏按钮大小
//设置按钮文本
m_wndToolBar.SetButtonText(0,"新建");
m_wndToolBar.SetButtonText(1,"打开");
m_wndToolBar.SetButtonText(2,"保存");
m_wndToolBar.SetButtonText(4,"剪切");
m_wndToolBar.SetButtonText(5,"复制");
m_wndToolBar.SetButtonText(6,"粘贴");
m_wndToolBar.SetButtonText(8,"打印");
m_wndToolBar.SetButtonText(10,"帮助");
```

(4) 在对话框的消息映射部分添加 TTN_NEEDTEXT 消息映射宏。

```
ON_NOTIFY_EX(TTN_NEEDTEXT, 0, OnToolTipNotify)
```

(5) 向对话框中添加 OnToolTipNotify 方法,代码如下:

```
BOOL CToolHintDlg::OnToolTipNotify(UINT id, NMHDR *pNMHDR, LRESULT *pResult)
{
    TOOLTIPTEXT *pTTT = (TOOLTIPTEXT *)pNMHDR;
    UINT nID = pNMHDR->idFrom; //获取工具栏按钮ID
    int index = m_wndToolBar.GetToolBarCtrl().CommandToIndex(nID); //根据ID获取按钮索引
    m_wndToolBar.GetButtonText(index, m_ToolText); //获取按钮文本
    pTTT->lpszText = m_ToolText.GetBuffer(0); //设置显示的提示信息
    pTTT->hinst = AfxGetResourceHandle();
    return(TRUE);
}
```

举一反三

根据本实例,读者可以:

- 实现具有提示功能的各种控件。

1.4 状态栏应用实例

状态栏位于主界面的底部，通常用于显示系统时间、程序运行时当前的状态信息等。在 MFC 中，提供了 CStatusBar 类用于创建和管理状态栏。本节将通过几个实例介绍状态栏的设计。

实例 015

使状态栏随对话框的改变而改变

这是一个可以提高分析能力的实例

实例位置：光盘\mingrisoft\01\015

实例说明

在对话框中创建一个状态栏，当窗口大小发生改变时，默认情况下，状态栏是不会发生改变的。但在实际应用中，需要状态栏随对话框大小改变而改变。效果如图 1.20 所示。

技术要点

对话框的大小发生变化后，对话框类的 OnSize 方法可以接收到消息，在 OnSize 方法内根据窗体改变后的大小来重新设置状态栏的大小。状态栏大小的改变使用 CStatusBar 类的 SetPaneInfo 方法实现。

SetPaneInfo 方法

该方法用来设置指定状态栏标识面板的 ID 属性值、样式和宽度。语法如下：

```
void SetPaneInfo( int nIndex, UINT nID, UINT nStyle, int cxWidth );
```

参数说明：

- nIndex：状态栏索引编号。
- nID：标识面板新资源 ID 值。
- nStyle：标识面板的样式。
- cxWidth：标识面板的宽度。

实现过程

- (1) 新建一个基于对话框的工程。
- (2) 在 CSizeStatusbarDlg 类中声明 CStatusBar 类对象。
- (3) 为 CSizeStatusbarDlg 类添加 WM_SIZE 消息的处理函数，在该函数中实现对状态栏大小的改变，函数实现代码如下：

```
void CSizeStatusbarDlg::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);
    if (IsWindow(m_StatusBar.m_hWnd)) //判断状态栏是否被创建
    {
        CRect rect;
        GetClientRect(rect); //获取客户端区域
        int width = rect.Width()/6;
        for (int i = 0; i < 6; i++)
        {
            m_StatusBar.SetPaneInfo(i, 1000+i, 0, width); //设置状态栏面板信息
        }
        RepositionBars(AFX_IDW_CONTROLBAR_FIRST, AFX_IDW_CONTROLBAR_LAST, 0);
    }
}
```

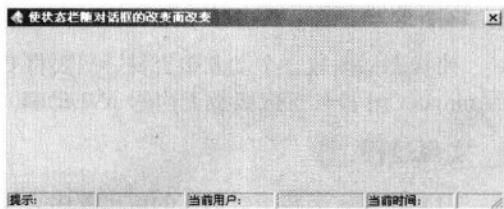


图 1.20 使状态栏随对话框的改变而改变

举一反三

根据本实例,读者可以:

- 实现具有组合框的状态栏。

实例 016 动画效果的状态栏

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\01\016

实例说明

在许多媒体软件中,状态栏中会播放一个动画,使界面更加美观。本实例就实现了一个动画效果的状态栏,效果如图 1.21 所示。



图 1.21 动画效果的状态栏

技术要点

使状态栏播放一个动画很容易,只要将 CAnimateCtrl 控件放置在状态栏中就可以了。因为 CAnimateCtrl 控件可以播放无声的 AVI 动画。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中放置 CAnimateCtrl 控件,并通过类向导将其命名为 m_Animate。
- (3) 在对话框类中定义一个 CStatusBar 变量 m_StatusBar。
- (4) 在对话框的 OnInitDialog 方法中创建状态栏,并将 CAnimateCtrl 控件显示在状态栏中。代码如下:

```
m_StatusBar.Create(this);           //创建状态栏
UINT  Indicates[4];
for (int i = 0; i < 4; i++)
{
    Indicates[i] = 50 + i;           //为数组元素赋值
}
m_StatusBar.SetIndicators(Indicates, 4); //设置面板ID
m_StatusBar.GetStatusBarCtrl().SetMinHeight(30); //设置面板高度
CRect rect;
GetClientRect(rect);                //获得窗体客户区域
UINT PaneWidth = rect.Width() / 5;  //计算面板宽度
for (int n = 0; n < 4; n++)
{
    m_StatusBar.SetPanelInfo(n, 50 + n * 10, SBPS_NORMAL, PaneWidth); //设置面板宽度
}
//设置状态栏面板文本
m_StatusBar.SetPaneText(0, "用户名称");
m_StatusBar.SetPaneText(1, "明日科技");
m_StatusBar.SetPaneText(2, "动画");
m_Animate.SetParent(&m_StatusBar); //设置动画控件的父窗口为状态栏
RepositionBars(AFX_IDW_CONTROLBAR_FIRST, AFX_IDW_CONTROLBAR_LAST, 0); //显示状态栏
CRect Rect;
m_StatusBar.GetStatusBarCtrl().GetRect(3, &Rect); //获得面板区域
CRect ProgRect(Rect.left, 2, Rect.right, Rect.Height() + 2); //设置显示位置
m_Animate.MoveWindow(ProgRect); //移动动画控件位置
m_Animate.Open("dmt.avi"); //打开AVI文件
m_Animate.Play(0, -1, -1); //播放AVI文件
```

举一反三

根据本实例,读者可以:

- 设计播放 FLASH 动画的状态栏。

实例 017 滚动字幕的状态栏

这是一个可以启发思维的实例

实例位置: 光盘\mingrisoft\01\017

实例说明

在火车站、客运站等许多公共场所, 随处可以看见一个大屏幕, 上面经常会以滚动字幕的形式显示一些信息。本实例实现了一个滚动字幕的状态栏, 效果如图 1.22 所示。

技术要点

在状态栏中实现滚动字幕, 可以利用静态文本控件实现。在状态栏中显示一个静态文本控件, 然后每隔一段时间调整静态文本控件的位置, 即可实现滚动字幕的效果了。

实现过程

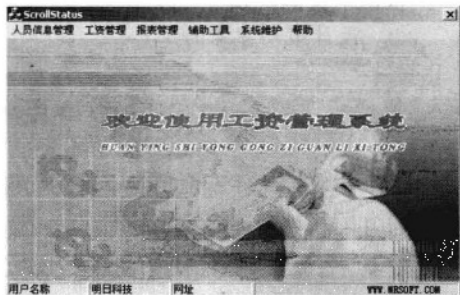


图 1.22 滚动字幕的状态栏

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中放置两个静态文本控件, 通过属性窗口设置控件的 ID 和 Caption 属性。
- (3) 通过类向导将两个静态文本控件分别命名为 m_Parent 和 m_Web。
- (4) 在对话框类的 OnInitDialog 方法中创建状态栏, 将静态文本控件显示在状态栏中, 代码如下:

```
m_StatusBar.Create(this);           //创建状态栏
UINT Indicates[4];
for (int i = 0; i<4;i++)
{
    Indicates[i] = 50+i;           //为数组元素赋值
}
m_StatusBar.SetIndicators(Indicates,4); //设置面板ID
CRect rect;
GetClientRect(rect);              //获得控件的客户区域
UINT PaneWidth = rect.Width()/6;   //计算面板宽度
//设置面板宽度
for(int n = 0;n<3;n++)
{
    m_StatusBar.SetPanelInfo(n,50+n*10,SBPS_NORMAL,PaneWidth); //设置面板宽度
}
//设置状态栏面板文本
m_StatusBar.SetPanelInfo(3,111,SBPS_NORMAL,800);
m_StatusBar.SetPaneText(0,"用户名称");
m_StatusBar.SetPaneText(1,"明日科技");
m_StatusBar.SetPaneText(2,"网址");
RepositionBars(AFX_IDW_CONTROLBAR_FIRST,AFX_IDW_CONTROLBAR_LAST,0); //显示状态栏
m_Parent.SetParent(&m_StatusBar); //设置静态文本控件的父窗口为状态栏
m_StatusBar.GetStatusBarCtrl().GetRect(3,rect); //获取控件的显示区域
rect.DeflateRect(1,1,1,1); //设置区域
m_Parent.MoveWindow(rect); //移动控件位置
m_Parent.GetClientRect(rect); //获得控件的客户区域
m_Web.GetClientRect(rect1); //获得控件的客户区域
m_Web.SetParent(&m_Parent); //设置控件的父窗口
m_Parent.GetClientRect(CurRect); //获得控件的客户区域
CurRect.DeflateRect(0,1,rect.Width()-rect1.Width(),1); //设置控件要移动到的区域
m_Web.MoveWindow(CurRect); //移动控件
SetTimer(1,200,NULL); //设置定时器
```

- (5) 处理对话框的 WM_TIMER 消息, 代码如下:

```
void CScrollStatusDlg::OnTimer(UINT nIDEvent)
{
    if (CurRect.left>=Rect.right) //如果移动区域左边大于等于面板右边界
    {
        CurRect.left = Rect.left-rect1.Width(); //设置移动区域左边界
    }
}
```



```

    CurRect.right = Rect.left;           //设置移动区域右边界
}
else                                   //否则
{
    CurRect.left += 4;                 //设置移动区域左边界+4
    CurRect.right += 4;               //设置移动区域右边界+4
}
//调整控件位置
m_Web.MoveWindow(CurRect);
}

```

举一反三

根据本实例，读者可以：

- 设计滚动字幕的工具栏。

1.5 导航界面应用实例

如今的应用软件不但具有丰富的功能，还应具有人性化的界面，这样，才能吸引用户。本节主要介绍几种常用的导航界面。

实例 018 Outlook 导航界面

本实例可以方便操作、提高效率

实例位置：光盘\mingrisoft\01\018

实例说明

采用导航式功能菜单，不但美观大方，而且为用户操作程序提供了方便。下面介绍 Outlook 导航界面式菜单的设计方法。运行程序，Outlook 式导航界面的效果如图 1.23 所示。

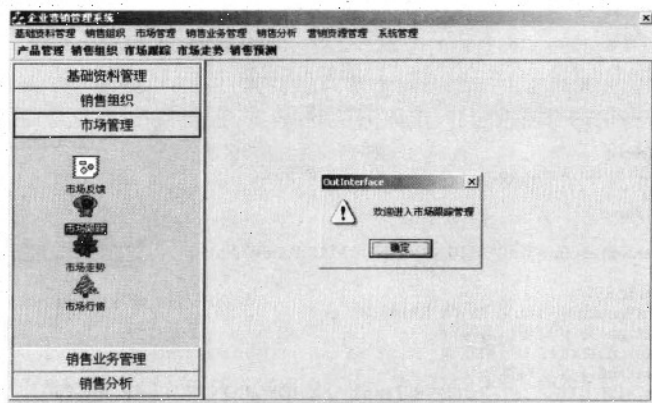


图 1.23 Outlook 导航界面

技术要点

为了设计 Outlook 导航界面，需要从 `CListCtrl` 派生一个子类，本实例为 `COutlookList`。在该类中显示一些导航按钮。当用户单击这些按钮时，会适当调整按钮的位置，并在客户区域（除按钮占用区域之外的区域）显示另一个 `CListCtrl` 控件，本实例为 `m_ClientList`，目的是显示与导航按钮关联的项目。在设计 `COutlookList` 类时，需要解决几个关键问题：一是如何截获导航按钮的单击事件，并确定用户单击了哪个按钮；二是如何存储与导航按钮关联的项目；三是如何向外界提供一个接口，以方便处理用户双击视图项执行的动作。

对于问题一，可以在 COutlookList 的 OnCmdMsg 方法中实现，代码如下：

```
BOOL COutlookList::OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO*
pHandlerInfo)
{
    int index = CommandToIndex(nID);
    if (index != -1)
    {
        OnButtonDown(index, nID);
    }
    m_ClientList.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
    return CListCtrl::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}
```

在 OnCmdMsg 方法中首先调用自定义的 CommandToIndex 方法获取命令对应的按钮索引，因为在创建导航按钮时，会为按钮指定 ID，并将按钮存储在 m_pButton 按钮数组中，只要遍历 m_pButton，就可以根据按钮 ID 确定索引了。如果导航按钮索引不为-1，表示用户单击了导航按钮，执行自定义的 OnButtonDown 方法，重新排列按钮，在客户区域显示列表控件。

对于问题二，可以从 CButton 派生一个子类（本实例为 CListButton），在该类中定义一个字符串列表 m_ButtonItems（类型为 CStringList），存储与导航按钮关联的项目文本。

对于问题三，可以定义一个回调函数，本实例为 ItemDlbFun，代码如下：

```
//定义双击列表视图项的回调函数
typedef void(ItemDlbFun)(const CListCtrl* pListCtrl, int nItemIndex);
```

然后在 COutlookList 类中定义一个 ItemDlbFun 函数指针 pItemDlbFun。最后在 COutlookList 类的 PreTranslateMessage 方法中判断用户是否双击了视图项，如果是，则调用 pItemDlbFun。

```
BOOL COutlookList::PreTranslateMessage(MSG* pMsg)
{
    if ((pMsg->hwnd==m_ClientList.m_hWnd)&&(pMsg->message==WM_LBUTTONDOWNBLCLK))
    {
        int index;
        index = m_ClientList.GetSelectionMark(); //获取用户双击的项目
        if (pItemDlbFun != NULL)
            pItemDlbFun(&m_ClientList, index);
    }
    return CListCtrl::PreTranslateMessage(pMsg);
}
```

实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框中添加列表视图控件和图片控件。设置图片控件属性如图 1.24 所示。

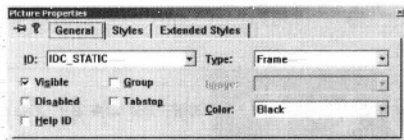


图 1.24 图片控件属性设置

(3) 从 CListCtrl 类派生一个子类 COutlookList，

在该类中定义如下成员变量：

CPtrArray m_pButton;	//按钮数组
UINT m_ButtonCount;	//按钮数量
UINT m_LeftMargin;	//图像列表显示的左边距
CListCtrl m_ClientList;	//在客户区域显示视图项
ItemDlbFun* pItemDlbFun;	//视图项的双击事件

(4) 从 CButton 类派生一个子类 CListButton，作为导航按钮。

(5) 在 CListButton 中定义如下成员变量：

UINT m_Index;	//导航按钮索引
BOOL m_Toped;	//按钮是否在列表上方
CStringList m_ButtonItems;	//按钮关联的图像列表项

(6) 在 COutlookList 中添加 AddButton 方法，用于添加导航按钮，代码如下：

```
void COutlookList::AddButton(LPCSTR Btntext, UINT uID)
{
    int index = m_pButton.Add(new CListButton);
    //创建按钮控件
    ((CListButton*)m_pButton[m_pButton.GetSize()-1])>>Create(Btntext, WS_CHILD, GetAddButtonRect(), this, uID);
    ((CListButton*)m_pButton[m_pButton.GetSize()-1])>>m_Index = index; //设置按钮索引
    ((CListButton*)m_pButton[m_pButton.GetSize()-1])>>ShowWindow(SW_SHOW); //显示控件
    m_ButtonCount++; //统计按钮数量
}
```

(7) 在 COutlookList 中添加 AddButtonItem 方法, 用于设置导航按钮关联的项目, 代码如下:

```
void COutlookList::AddButtonItem(UINT nIndex, CString &strItem)
{
    CListButton* temp;
    temp = (CListButton*)m_pButton[nIndex];           //获得按钮指针
    temp->m_ButtonItems.AddTail(strItem);              //记录列表项
}
```

(8) 在 COutlookList 中添加 ShowButtonItem 方法, 用于显示指定导航按钮关联的项目, 代码如下:

```
void COutlookList::ShowButtonItem(UINT nIndex)
{
    CListButton* temp;
    temp = (CListButton*)m_pButton[nIndex];           //获取指定的导航按钮
    m_ClientList.DeleteAllItems();                     //清空列表控件
    CRect showrect = GetListClientRect();             //获取客户区域
    if (temp->m_ButtonItems.GetCount()>0)
    {
        POSITION pos;
        pos = temp->m_ButtonItems.GetHeadPosition();   //列表项索引
        CString str = temp->m_ButtonItems.GetHead();   //获得列表项文本
        CRect ClientRect;
        ClientRect = GetListClientRect();              //获得列表区域
        int m = 0;
        m_ClientList.InsertItem(m, str, m);            //插入列表项
        m_ClientList.SetItemPosition(m, CPoint(m_LeftMargin, 20+m*48)); //设置列表项位置
        while (pos != temp->m_ButtonItems.GetTailPosition())
        {
            str = temp->m_ButtonItems.GetNext(pos);     //获得下一个列表项文本
            m_ClientList.InsertItem(m, str, m);         //插入列表项
            m_ClientList.SetItemPosition(m, CPoint(m_LeftMargin, 20+m*48)); //设置列表项位置
            m+=1;
        }
        str = temp->m_ButtonItems.GetAt(pos);           //获得文本
        m_ClientList.InsertItem(m, str, m);            //插入列表项
        m_ClientList.SetItemPosition(m, CPoint(m_LeftMargin, 20+m*48)); //设置列表项位置
    }
}
```

举一反三

根据本实例, 读者可以:

- 利用 TreeControl 控件制作导航界面。

实例 019 树状导航界面

本实例可以美化界面、简化操作

实例位置: 光盘\mingrisoft\01\019

实例说明

系统中的资源浏览器左边就是一个树状的导航界面。本实例利用两层树型视图来实现树状导航界面, 用户可以通过双击树型视图的单个项实现某些功能, 如图 1.25 所示。

技术要点

本实例通过窗体的分割技术将树型视图显示在左边, 然后通过树型视图的 GetTreeCtrl 方法来获得 CTreeCtrl 对象, 通过 CTreeCtrl 对象接受双击事件来实现导航功能。

实现过程

- (1) 新建名为 TreeNavi 的单文档应用程序。



图 1.25 树状导航界面

(2) 在工程中添加以 CTreeView 类为基类的新类 TreeView，在工程中添加 Icon 资源。

(3) 在 TreeView.h 文件中添加如下代码：

```
#include "afxview.h"
```

(4) 修改 TreeNavi.cpp 文件中 InitInstance 函数的内容。代码如下：

```
BOOL CTreeViewApp::InitInstance()
{
    ...//此处代码省略
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CTreeViewDoc),
        RUNTIME_CLASS(CMainFrame),
        //去除原有的TreeNaviView视图的应用
        //    RUNTIME_CLASS(CTreeViewView)
        NULL);
    AddDocTemplate(pDocTemplate);
    ...//此处代码省略
    return TRUE;
}
```

(5) 在 MainFrm.cpp 文件中添加 WM_CREATECLIENT 消息的实现函数，实现分割窗体，代码如下：

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
{
    split.CreateStatic(this,1,2);
    split.CreateView(0,0,RUNTIME_CLASS(TreeView),CSize(100,100),pContext); //创建分割窗体
    split.CreateView(0,1,RUNTIME_CLASS(CTreeViewView),CSize(100,100),pContext); //创建左侧子视图
    return CFrameWnd::OnCreateClient(lpcs, pContext); //创建右侧子视图
}
```

(6) 函数 OnInitialUpdate 实现树型视图的初始化，代码如下：

```
void CTreeView::OnInitialUpdate()
{
    CTreeView::OnInitialUpdate();
    list.Create(32,32,ILC_COLOR32|ILC_MASK,0,0); //创建图像列表
    list.Add(::AfxGetApp()->LoadIcon(IDI_ICON1)); //加载图标资源
    ...//此处代码省略
    this->GetTreeCtrl().SetImageList(&list,TVSIL_NORMAL); //关联图像列表
    HTREEITEM tree;
    //创建树状结构
    tree=this->GetTreeCtrl().InsertItem("导航1",0,1);
    this->GetTreeCtrl().InsertItem("子导航1",6,6,tree);
    this->GetTreeCtrl().InsertItem("子导航2",7,7,tree);
    this->GetTreeCtrl().InsertItem("子导航3",8,8,tree);
    tree=this->GetTreeCtrl().InsertItem("导航2",2,3);
    this->GetTreeCtrl().InsertItem("子导航4",9,9,tree);
    this->GetTreeCtrl().InsertItem("子导航5",10,10,tree);
    this->GetTreeCtrl().InsertItem("子导航6",11,11,tree);
    tree=this->GetTreeCtrl().InsertItem("导航3",4,5);
    this->GetTreeCtrl().InsertItem("子导航7",12,12,tree);
    this->GetTreeCtrl().InsertItem("子导航8",13,13,tree);
    this->GetTreeCtrl().InsertItem("子导航9",14,14,tree);
}
```

(7) 添加 NM_DBLCLK 消息的实现函数 OnDbclclk，代码如下：

```
void CTreeView::OnDbclclk(NMHDR* pNMHDR, LRESULT* pResult)
{
    HTREEITEM tree=this->GetTreeCtrl().GetSelectedItem();
    if(!this->GetTreeCtrl().GetChildItem(tree))
    {
        CString str=this->GetTreeCtrl().GetItemText(tree); //获得树节点文本
        AfxMessageBox(str); //弹出消息提示
    }
    *pResult = 0;
}
```

举一反三

根据本实例，读者可以：

- 实现列表视图导航。

实例 020 按钮导航界面

本实例可以美化界面、简化操作

实例位置：光盘\mingrisoft\01\020

实例说明

一般应用程序都使用菜单作为导航，用户通过操作菜单来实现某些功能。本实例使用多个按钮来作为用户操作的引导，效果如图 1.26 所示。

技术要点

为了使界面比较漂亮，本实例以 CButton 类为基类派生一个子类 CCustomButton，然后声明一个枚举变量，用于设置按钮控件的 3 种状态，分别是正常状态、热点状态、按下状态，在 CCustomButton 类的 WM_MOUSEMOVE 事件中设置按钮是否为热点状态，在 WM_LBUTTONDOWN 事件和 WM_LBUTTONUP 事件中设置按钮的按下和正常状态，最后在 CCustomButton 类的 OnPaint 方法中绘制按钮，这样就使程序界面看起来更加美观。

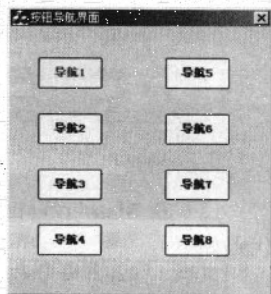


图 1.26 按钮导航界面

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加两个按钮控件。
- (3) 以 CButton 类为基类派生一个按钮类 CCustomButton，并为两个按钮控件分别关联一个 CCustomButton 类的变量。

- (4) 定义一个枚举类型，代码如下：

```
enum ButtonState {bsNormal,bsHot,bsDown}; //正常状态，热点状态、按下状态
```

- (5) 在 CCustomButton 类的头文件中声明变量，代码如下：

```
int m_State; //按钮当前状态
```

- (6) 在 CCustomButton 类的构造函数和析构函数中设置按钮的状态，代码如下：

```
CCustomButton::CCustomButton() //构造函数
{
    m_State = bsNormal; //设置为默认状态
}
CCustomButton::~~CCustomButton() //析构函数
{
    m_State = bsNormal; //设置为默认状态
}
```

- (7) 在 CCustomButton 类中，处理按钮的 WM_MOUSEMOVE 事件，在该事件的处理函数中设置按钮是否为热点状态，代码如下：

```
void CCustomButton::OnMouseMove(UINT nFlags, CPoint point)
{
    HRGN hRgn = CreateRectRgn(0, 0, 0, 0);
    GetWindowRgn(hRgn); //获得按钮区域
    BOOL ret = PtInRegion(hRgn, point.x, point.y); //鼠标是否在按钮上
    if(ret) //在按钮上
    {
        if(m_State == bsDown) //判断按钮是否为按下状态
            return;
        if(m_State != bsHot) //判断按钮是否为热点状态
        {
            m_State = bsHot; //设置为热点状态
            InvalidateRect(NULL, TRUE); //更新按钮
            SetCapture(); //捕获鼠标
        }
    }
    else //不在按钮上
```

```
{
    m_State = bsNormal;           //设置按钮状态
    InvalidateRect(NULL,TRUE);    //更新按钮
    ReleaseCapture();             //释放鼠标
}
DeleteObject( hRgn );
CButton::OnMouseMove(nFlags, point);
}
```

(8) 在 CCustomButton 类中, 处理按钮的 WM_LBUTTONDOWN 事件, 在该事件的处理函数中设置按钮为按下状态, 代码如下:

```
void CCustomButton::OnLButtonDown(UINT nFlags, CPoint point)
{
    m_State = bsDown;           //设置按钮按下状态
    InvalidateRect(NULL,TRUE);  //更新按钮
}
```

(9) 在 CCustomButton 类中, 处理按钮的 WM_LBUTTONUP 事件, 在该事件的处理函数中设置按钮为默认状态, 代码如下:

```
void CCustomButton::OnLButtonUp(UINT nFlags, CPoint point)
{
    if(m_State != bsNormal)     //判断按钮状态
    {
        m_State = bsNormal;     //设置按钮状态
        ReleaseCapture();       //释放鼠标捕捉
        InvalidateRect(NULL,TRUE); //更新按钮
    }
    //向父窗口发送命令消息
    ::SendMessage(GetParent()->m_hWnd,WM_COMMAND, GetDlgCtrlID(), (LPARAM) m_hWnd);
}
```

(10) 在 CCustomButton 类中, 处理按钮的 WM_PAINT 事件, 在该事件的处理函数中根据按钮的状态绘制按钮控件, 代码如下:

```
void CCustomButton::OnPaint()
{
    CPaintDC dc(this);          //获取按钮的设备上下文
    CString Text;               //定义一个字符串变量
    CRect RC;                   //定义一个区域对象
    CFont Font;                 //定义一个字体对象
    CFont *pOldFont;            //定义一个字体对象指针, 用于存储之前的字体
    CBrush Brush;               //定义一个画刷对象
    CBrush *pOldBrush;          //定义一个画刷对象指针, 用于存储之前的画刷对象
    CPoint PT(2,2);             //定义一个点对象
    dc.SetBkMode(TRANSPARENT); //将设备上下文背景模式设置为透明
    Font.CreateFont(12, 0, 0, 0, FW_HEAVY, 0, 0, 0, ANSI_CHARSET,
        OUT_TT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
        VARIABLE_PITCH | FF_SWISS, "宋体"); //创建字体
    pOldFont = dc.SelectObject(&Font); //选中新的字体
    if(m_State == bsNormal)      //判断按钮是否为正常状态
    {
        Brush.CreateSolidBrush( RGB(230, 230, 230)); //创建指定颜色的画刷
        dc.SetTextColor(RGB(0, 0, 0)); //设置文本颜色
    }
    else if(m_State == bsDown)  //判断按钮是否为按下状态
    {
        Brush.CreateSolidBrush(RGB(100, 100, 180)); //创建指定颜色的画刷
        dc.SetTextColor(RGB(250, 250, 0)); //设置文本颜色
    }
    else if(m_State == bsHot)   //判断按钮是否为热点状态
    {
        Brush.CreateSolidBrush(RGB(230, 230, 130)); //创建指定颜色的画刷
        dc.SetTextColor(RGB(50, 50, 250)); //设置文本颜色
    }
    pOldBrush = dc.SelectObject(&Brush); //选中画刷
    GetClientRect(&RC); //获取按钮的客户区域
    dc.RoundRect(&RC, PT); //利用当前选中的画刷和画笔绘制按钮区域
    HRGN hRgn = CreateRectRgn(RC.left, RC.top, RC.right, RC.bottom); //创建一个选区
    SetWindowRgn(hRgn, TRUE); //设置按钮窗口区域
    DeleteObject(hRgn); //删除选区
    GetWindowText( Text); //获取按钮文本
    dc.DrawText(Text, &RC, DT_CENTER | DT_VCENTER | DT_SINGLELINE); //绘制按钮文本
    Font.DeleteObject(); //删除字体对象
    Brush.DeleteObject(); //删除画刷对象
    dc.SelectObject(pOldFont); //恢复原来选中的字体
    dc.SelectObject(pOldBrush); //恢复原来选中的画刷
}
```

举一反三

根据本实例,读者可以:

- 在视图中利用按钮导航。

实例 021 图片导航界面

本实例可以美化界面、简化操作

实例位置: 光盘\mingrisoft\01\021

实例说明

本实例实现了图片导航界面。程序中的每个按钮中都有一个图片,当用户的鼠标在图片上滑过时,按钮显示热点效果的图片,使用户清楚地知道当前执行的操作,通过单击不同的图片可以实现相应的功能。程序运行效果如图 1.27 所示。

技术要点

本实例主要通过按钮控件的显示和隐藏来实现,首先设置按钮控件全部隐藏,然后在鼠标滑过当前按钮控件时为按钮控件设置显示图片,并显示当前的按钮控件,从而达到控件的热点效果。

实现过程



图 1.27 图片导航界面

(1) 新建名为 Navigation 的对话框应用程序。

(2) 在工程中添加 5 个位图文件,并添加一个图片控件和 4 个按钮控件,设置按钮控件的 Bitmap 属性和 Flat 属性,设置图片控件显示背景位图。

(3) OnInitDialog 函数中设置按钮控件隐藏,代码如下:

```
BOOL CNavigationDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    //系统代码省略
    //隐藏按钮控件
    GetDlgItem(IDC_BUTTON1)->ShowWindow(SW_HIDE);
    GetDlgItem(IDC_BUTTON2)->ShowWindow(SW_HIDE);
    GetDlgItem(IDC_BUTTON3)->ShowWindow(SW_HIDE);
    GetDlgItem(IDC_BUTTON4)->ShowWindow(SW_HIDE);
    return TRUE;
}
```

(4) 处理主窗体的 WM_MOUSEMOVE 消息,在该消息的处理函数中设置按钮的显示图片,并显示指定按钮控件,代码如下:

```
void CNavigationDlg::OnMouseMove(UINT nFlags, CPoint point)
{
    int x = point.x;
    int y = point.y;
    if (x>=27 && y>=141 && x<=56 && y<=171) //判断鼠标是否在基本信息按钮范围内
    {
        HBITMAP hbmp;
        hbmp = (HBITMAP)::LoadImage(AfxFindResourceHandle(MAKEINTRESOURCE(IDB_BITMAP1),
            RT_GROUP_ICON),MAKEINTRESOURCE(IDB_BITMAP1), IMAGE_BITMAP, 0, 0, 0); //加载图片资源
        CButton *p = (CButton *)GetDlgItem(IDC_BUTTON1); //获得按钮指针
        p->ShowWindow(SW_SHOW); //显示控件
        p->SetFocus(); //按钮控件获得焦点
        p->SetBitmap(hbmp); //显示图片
    }
    else if (x>=27 && y>=192 && x<=56 && y<=220) //判断鼠标是否在库存管理按钮范围内
```



```
{
    HBITMAP hbmp;
    hbmp = (HBITMAP)::LoadImage(AfxFindResourceHandle(MAKEINTRESOURCE(IDB_BITMAP2),
        RT_GROUP_ICON),MAKEINTRESOURCE(IDB_BITMAP2), IMAGE_BITMAP, 0, 0, 0); //加载图片资源
    CButton *p = (CButton *)GetDlgItem(IDC_BUTTON2); //获得按钮指针
    p->ShowWindow(SW_SHOW); //显示控件
    p->SetBitmap(hbmp); //显示图片
}
else if(x>=27 && y>=245 && x<=56 && y<=266) //判断鼠标是否在查询管理按钮范围内
{
    HBITMAP hbmp;
    hbmp = (HBITMAP)::LoadImage(AfxFindResourceHandle(MAKEINTRESOURCE(IDB_BITMAP3),
        RT_GROUP_ICON),MAKEINTRESOURCE(IDB_BITMAP3), IMAGE_BITMAP, 0, 0, 0); //加载图片资源
    CButton *p = (CButton *)GetDlgItem(IDC_BUTTON3); //获得按钮指针
    p->ShowWindow(SW_SHOW); //显示控件
    p->SetBitmap(hbmp); //显示图片
}
else if(x>=27 && y>=289 && x<=56 && y<=318) //判断鼠标是否在系统设置按钮范围内
{
    HBITMAP hbmp;
    hbmp = (HBITMAP)::LoadImage(AfxFindResourceHandle(MAKEINTRESOURCE(IDB_BITMAP4),
        RT_GROUP_ICON),MAKEINTRESOURCE(IDB_BITMAP4), IMAGE_BITMAP, 0, 0, 0); //加载图片资源
    CButton *p = (CButton *)GetDlgItem(IDC_BUTTON4); //获得按钮指针
    p->ShowWindow(SW_SHOW); //显示控件
    p->SetBitmap(hbmp); //显示图片
}
else
{
    //隐藏按钮控件
    GetDlgItem(IDC_BUTTON1)->ShowWindow(SW_HIDE);
    GetDlgItem(IDC_BUTTON2)->ShowWindow(SW_HIDE);
    GetDlgItem(IDC_BUTTON3)->ShowWindow(SW_HIDE);
    GetDlgItem(IDC_BUTTON4)->ShowWindow(SW_HIDE);
}
CDialog::OnMouseMove(nFlags, point);
}
```

举一反三

根据本实例，读者可以：

- 开发图标导航界面。

1.6 界面窗体应用实例

在设计程序界面时，首先考虑的是对话框的设计。本节通过几个实例介绍各种对话框程序界面效果的设计。

实例 022 使用位图设计畸形界面

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\01\022

实例说明

在开发应用程序时为使程序界面更加美观，可以将程序界面设计成各种不规则图案。本实例实现了使用位图设计畸形界面。运行程序，程序的背景由位图的图案进行设置，然后再将位图画到程序背景上。程序运行效果如图 1.28 所示。

技术要点

利用位图设计不规则窗体的主要思路是把想挖去的区域设成黑色，然后利用 GetPixe 方法挖去背景色，将不是背景色的区域用 CombineRgn 方法连接起来，最后用 SetWindowRgn 方法设置窗体区域。

CombineRgn 方法用于设置一个 CRgn 对象，使它等效于两个指定的 CRgn 对象的联合。语



图 1.28 使用位图设计畸形界面

法如下:

```
int CombineRgn( CRgn* pRgn1, CRgn* pRgn2, int nCombineMode );
```

参数说明:

- pRgn1: 一个已经存在的区域。
- pRgn2: 一个已经存在的区域。
- nCombineMode: 组合两个源区域时要执行的操作。
 - RGN_AND: 使用两个区域相互重叠的区域, 即相交的部分。
 - RGN_COPY: 创建参数 pRgn1 标识的区域的一个备份。
 - RGN_DIFF: 创建一个区域, 该区域由区域 1 (pRgn1 标识的区域) 去除在区域 2 (pRgn2 标识的区域) 中的部分而形成的区域。
 - RGN_OR: 组合两个区域的所有部分。
 - RGN_XOR: 组合两个区域, 去除相互重叠的区域。

实现过程

- (1) 新建名为 Catface 的对话框应用程序。
- (2) 向工程中添加一个位图资源。
- (3) 在对话框的 OnPaint 方法中设置窗体区域, 代码如下:

```
void CCatfaceDlg::OnPaint()
{
    //系统代码省略
    CDC *pDC = GetDC(); //获得设备上下文
    CDC memDC;
    CBitmap bitmap; //声明位图对象
    CBitmap* bmp = NULL;
    COLORREF col;
    CRect rc;
    int x, y;
    CRgn rgn, tmp;
    GetWindowRect(&rc); //获得窗体区域
    bitmap.LoadBitmap(IDB_BITMAP1); //装载模板位图
    memDC.CreateCompatibleDC(pDC); //创建与内存兼容的设备上下文
    bmp = memDC.SelectObject(&bitmap);
    rgn.CreateRectRgn(0, 0, rc.Width(), rc.Height()); //初始化区域
    //计算得到区域
    for(x=0; x<=rc.Width(); x++)
    {
        for(y=0; y<=rc.Height(); y++)
        {
            //将背景部分去掉
            col = memDC.GetPixel(x, y); //得到像素颜色
            if(col == RGB(255,255,255)) //如果是背景颜色
            {
                tmp.CreateRectRgn(x, y, x+1, y+1); //创建区域
                rgn.CombineRgn(&rgn, &tmp, RGN_XOR); //去除相互重叠的区域
                tmp.DeleteObject(); //删除区域对象
            }
        }
    }
    SetWindowRgn((HRGN)rgn, TRUE); //设置窗体为区域的形状
}
```

- (4) 处理窗体的 WM_CTLCOLOR 消息, 在该消息的处理函数中绘制窗体背景位图, 代码如下:

```
HBRUSH CCatfaceDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
    CBitmap m_BKGround;
    m_BKGround.LoadBitmap(IDB_BITMAP1); //加载图片资源
    if (nCtlColor==CTL_COLOR_DLG)
    {
        CBrush m_Brush(&m_BKGround); //定义一个位图画刷
        CRect rect;
        GetClientRect(rect); //获得窗体的客户区域
        pDC->SelectObject(&m_Brush); //选中画刷
        pDC->FillRect(rect, &m_Brush); //填充客户区域
        return m_Brush;
    }
}
```

```

}
else
    hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
return hbr;
}

```

举一反三

根据本实例，读者可以：

- 实现菜单背景颜色的渐变；
- 实现工具栏背景颜色的渐变。

实例 023 制作立体窗口阴影效果

本实例可以美化界面、简化操作

实例位置：光盘\mingrisoft\01\023

实例说明

制作立体窗口阴影效果是在主窗口的右边和下边放两个非模态对话框，然后半透明显示，就形成了阴影效果。在 WM_MOVE 消息的处理函数中设置非模态对话框的显示位置，如图 1.29 所示。

技术要点

为了实现阴影效果，需要将作为阴影的窗体半透明。调用 User32 动态库中的 SetLayeredWindowAttributes 函数设置半透明窗体。在 Visual C++ 中，SetLayeredWindowAttributes 函数并没有被直接封装，需要用户手工从 User32 动态库中导入。

使窗体具有 0x80000 值的扩展风格很容易，可以调用 SetWindowLong API 函数实现。该函数语法如下：

```
LONG SetWindowLong(HWND hWnd, int nIndex, LONG dwNewLong);
```

参数说明：

- hWnd：表示窗口句柄。
- nIndex：表示修改窗口的哪一特征，本例需要修改窗口的扩展风格，因此该参数应为 GWL_EXSTYLE。
- dwNewLong：表示窗口新的特征。

导入 SetLayeredWindowAttributes 函数，首先需要定义一个与 SetLayeredWindowAttributes 函数具有相同函数原型的函数指针，例如：

```
typedef BOOL (WINAPI *FSetLayeredWindowAttributes)(HWND, COLORREF, BYTE, DWORD);
FSetLayeredWindowAttributes SetLayeredWindowAttributes;
```

然后调用 LoadLibrary 函数加载 User32 动态库，最后调用 GetProcAddress 函数将 SetLayeredWindowAttributes 指向 User32 动态库中的 SetLayeredWindowAttributes 函数。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 添加两个对话框窗体类 CShaddlg1 和 Cshaddlg2。在对话框的 OnInitDialog 方法中实现窗体的半透明效果。代码如下：

```

SetWindowLong(GetSafeHwnd(), GWL_EXSTYLE,
    GetWindowLong(GetSafeHwnd(), GWL_EXSTYLE) | 0x80000); //设置窗口扩展风格
typedef BOOL (WINAPI *FSetLayeredWindowAttributes)(HWND, COLORREF, BYTE, DWORD); //定义函数指针类型
FSetLayeredWindowAttributes SetLayeredWindowAttributes; //定义函数指针
HINSTANCE hInst = LoadLibrary("User32.DLL"); //载入DLL
SetLayeredWindowAttributes = (FSetLayeredWindowAttributes)

```

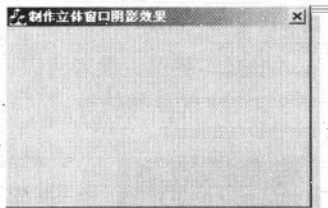


图 1.29 制作立体窗口阴影效果

```
GetProcAddress(hInst, "SetLayeredWindowAttributes"); //获取函数指针地址
if(SetLayeredWindowAttributes)
SetLayeredWindowAttributes(GetSafeHwnd(), RGB(0,0,0), 128, 1); //设置透明
FreeLibrary(hInst); //释放DLL
```

(3) 在主对话框的 OnCreate 方法中创建作为阴影的窗体。代码如下:

```
int CShadowDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if(CDialog::OnCreate(lpCreateStruct) == -1)
        return -1;
    dlg1.Create(IDD_DIALOG1, this);
    dlg2.Create(IDD_DIALOG2, this);
    return 0;
}
```

(4) 在主对话框移动时修改阴影窗体的位置。代码如下:

```
void CShadowDlg::OnMove(int x, int y)
{
    CDialog::OnMove(x, y);
    CRect rect;
    GetWindowRect(&rect);
    dlg1.MoveWindow(rect.left+10, rect.bottom, rect.Width()-10, 10); //获取主窗体大小
    //移动下阴影窗体
    dlg1.ShowWindow(SW_SHOW); //显示窗体
    dlg2.MoveWindow(rect.right, rect.top+10, 10, rect.Height()); //移动右阴影窗体
    //显示窗体
    dlg2.ShowWindow(SW_SHOW);
}
```

举一反三

根据本实例, 读者可以:

- 实现各种特殊形状的界面添加阴影效果。

实例 024 自绘窗体界面

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\01\024

实例说明

如今的软件不仅具有强大的功能, 还具有漂亮的界面, 这些漂亮的界面大大增加了用户使用程序的乐趣。例如, 大家上网经常使用的瑞星、腾讯 OICQ 等软件。本实例实现了一个自绘的窗体界面, 效果如图 1.30 所示。

技术要点

要实现窗体的绘制并不像想象中那么复杂。首先需要准备几个漂亮的位图, 用于作为窗体的边框和标题栏, 然后利用设备上下文 CDC 将其绘制在窗体上就可以了。CDC 提供了 StretchBlt 方法, 用于绘制图像。其语法如下:

```
BOOL StretchBlt(int x, int y, int nWidth, int nHeight, CDC* pSrcDC, int xSrc, int ySrc, int nSrcWidth, int nSrcHeight, DWORD dwRop);
```

参数说明:

- x、y: 表示目标区域的左上角坐标。
- nWidth、nHeight: 表示目标区域的宽度和高度。
- pSrcDC: 表示源设备上下文指针。
- xSrc、ySrc: 表示源设备上下文的左上角坐标。
- nSrcWidth、nSrcHeight: 表示源设备上下文的宽度和高度。
- dwRop: 表示光栅效果。

绘制窗体的标题栏和边框时不能使用 GetDC 方法获得设备上下文指针, 因为 GetDC 方法获得的是窗体客户区域的设备上下文指针。应使用 GetWindowDC 方法获得窗口设备上下文指针。

本实例在标题栏上利用位图绘制了几个按钮, 当用户单击不同的按钮时, 会执行相应的操作。

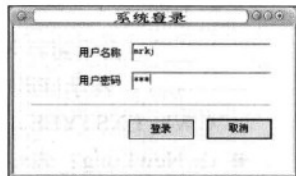


图 1.30 自绘窗体界面

在绘制标题栏按钮时,需要知道标题栏按钮的显示区域(该区域由用户根据位图的大小适当设置),该区域会随着窗体的大小变化而不同。知道了按钮的显示区域,当用户鼠标在标题栏上移动时,判断鼠标点是否在按钮区域,如果在,将按钮状态 `m_ButtonState` (是一个枚举值) 设置为相应的值。然后再处理鼠标左键在非客户区域按下时的消息,在消息处理函数中判断 `m_ButtonState` 值,根据不同的值执行相应的动作。这样便实现了标题栏按钮的单击事件。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加编辑框、静态文本、按钮和图片控件。
- (3) 在对话框中添加如下成员变量。

<code>BOOL m_IsMax;</code>	//是否处于最大化状态
<code>int m_BorderWidth;</code>	//边框宽度
<code>int m_BorderHeight;</code>	//边框高度
<code>int m_CaptionHeight;</code>	//标题栏的高度
<code>CString m_Caption;</code>	//窗口标题
<code>COLORREF m_CapitonColor;</code>	//标题字体颜色
<code>CFont m_CaptionFont;</code>	//标题字体
<code>int m_ButtonWidth;</code>	//按钮位图宽度
<code>int m_ButtonHeight;</code>	//按钮位图高度
<code>BOOL m_FirstShow;</code>	//窗口首次被显示
<code>CRect m_OrigionRect;</code>	//原始窗口区域
<code>CRect m_IniRect,m_MinRect,m_MaxRect,m_CloseRect;</code>	//标题栏按钮的显示区域
<code>CButtonState m_ButtonState;</code>	//按钮状态
<code>BOOL m_IsDrawForm;</code>	//是否需要绘制窗体

- (4) 在对话框中添加 `DrawForm` 方法绘制窗体,代码如下:

```
void CDrawFormDlg::DrawForm()
{
    CDC* pWindowDC = GetWindowDC();           //获取窗口设备上下文
    CBitmap LeftLine;
    BITMAPINFO bitinfo;
    CDC memDC;
    memDC.CreateCompatibleDC(pWindowDC);
    CRect Clientrect;
    GetClientRect(Clientrect);                 //获得窗体客户区域
    int leftwidth=0;                            //左标题的宽度
    int rightwidth = 0;                         //右标题的宽度
    int leftlinewidth = 0;                      //左边线宽度
    LeftLine.LoadBitmap(IDB_BITMAP3);           //加载右标题
    LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获得位图大小
    rightwidth = bitinfo.bmiHeader.biWidth;     //获得宽度
    LeftLine.DeleteObject();
    int x,y;
    //绘制左边线
    //获取位图大小
    LeftLine.LoadBitmap(IDB_BITMAP4);           //加载位图资源
    LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获得位图大小
    leftlinewidth = x = bitinfo.bmiHeader.biWidth; //获得宽度
    y = bitinfo.bmiHeader.biHeight;             //获得高度
    memDC.SelectObject(&LeftLine);              //选入图片
    pWindowDC->StretchBlt(1-m_BorderWidth,m_CaptionHeight+1,x+1,Clientrect.Height()
        +2*m_BorderHeight+5,memDC,0,0,x,y,SRCCOPY); //绘制图片
    LeftLine.DeleteObject();

    /*****绘制左标题*****/
    LeftLine.LoadBitmap(IDB_BITMAP2);           //加载图片资源
    LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获取位图大小
    memDC.SelectObject(&LeftLine);              //选入位图
    leftwidth = x = bitinfo.bmiHeader.biWidth;  //获得宽度
    y = bitinfo.bmiHeader.biHeight;             //获得高度
    pWindowDC->StretchBlt(-m_BorderWidth,0,x,m_CaptionHeight+4,memDC,0,0,x,y,SRCCOPY); //绘制图片
    LeftLine.DeleteObject();
    /*****绘制左标题*****/

    /*****绘制中间标题*****/
    LeftLine.LoadBitmap(IDB_BITMAP1);           //加载图片资源
    LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获取位图大小
    memDC.SelectObject(&LeftLine);              //选入图片
    x = bitinfo.bmiHeader.biWidth;              //获得宽度
```



```

y = bitinfo.bmiHeader.biHeight; //获得高度
pWindowDC->StretchBlt(leftwidth-1,0,Clientrect.Width()-leftwidth-rightwidth,m_
CaptionHeight+4,&memDC,0,0,x,y,SRCCOPY); //绘制图片
LeftLine.DeleteObject();
/*****绘制中间标题*****/

/*****绘制右标题*****/
LeftLine.LoadBitmap(IDB_BITMAP3); //加载图片资源
LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获取位图大小
memDC.SelectObject(&LeftLine); //选入位图
x = bitinfo.bmiHeader.biWidth; //获得宽度
y = bitinfo.bmiHeader.biHeight; //获得高度
pWindowDC->StretchBlt(Clientrect.Width()-x-1,0,x+m_BorderWidth+9
,m_CaptionHeight+4,&memDC,0,0,x,y,SRCCOPY); //绘制图片
LeftLine.DeleteObject();
/*****绘制右标题*****/

/*****绘制右边框*****/
LeftLine.LoadBitmap(IDB_BITMAP4); //加载图片资源
LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获取位图大小
memDC.SelectObject(&LeftLine); //选入位图
x = bitinfo.bmiHeader.biWidth; //获得位图宽度
y = bitinfo.bmiHeader.biHeight; //获得位图高度
pWindowDC->StretchBlt(Clientrect.Width()+m_BorderWidth+2,m_CaptionHeight+1,
x+m_BorderWidth,Clientrect.Height()+2*m_BorderHeight+5,&memDC,0,0,x,y,SRCCOPY); //绘制图片
LeftLine.DeleteObject();
/*****绘制右边框*****/

/*****绘制底边框*****/
LeftLine.LoadBitmap(IDB_BITMAP5); //加载图片资源
LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获取位图大小
memDC.SelectObject(&LeftLine); //选入图片
x = bitinfo.bmiHeader.biWidth; //获得宽度
y = bitinfo.bmiHeader.biHeight; //获得高度
pWindowDC->StretchBlt(leftlinewidth-m_BorderWidth,Clientrect.Height()
+m_CaptionHeight+2,Clientrect.Width()-m_BorderWidth,y+2,&memDC,0,0,x,y,SRCCOPY); //绘制图片
LeftLine.DeleteObject();
/*****绘制底边框*****/

/*****绘制初始化按钮*****/
LeftLine.LoadBitmap(IDB_BITMAP6); //加载图片资源
LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获取位图大小
memDC.SelectObject(&LeftLine); //选入位图
x = bitinfo.bmiHeader.biWidth; //获得宽度
y = bitinfo.bmiHeader.biHeight; //获得高度
pWindowDC->StretchBlt(m_IniRect.left,m_IniRect.top,m_IniRect.right
,m_IniRect.bottom,&memDC,0,0,x,y,SRCCOPY); //绘制图片
LeftLine.DeleteObject();
/*****绘制初始化按钮*****/

/*****绘制最小化按钮*****/
LeftLine.LoadBitmap(IDB_BITMAP6); //加载图片资源
LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获取位图大小
memDC.SelectObject(&LeftLine); //选入位图
x = bitinfo.bmiHeader.biWidth; //获得宽度
y = bitinfo.bmiHeader.biHeight; //获得高度
pWindowDC->StretchBlt(m_MinRect.left,m_MinRect.top,m_MinRect.right,
m_MinRect.bottom,&memDC,0,0,x,y,SRCCOPY); //绘制图片
LeftLine.DeleteObject();
/*****绘制最小化按钮*****/

/*****绘制最大化按钮*****/
LeftLine.LoadBitmap(IDB_BITMAP6); //加载图片资源
LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获取位图大小
memDC.SelectObject(&LeftLine); //选入位图
x = bitinfo.bmiHeader.biWidth; //获得宽度
y = bitinfo.bmiHeader.biHeight; //获得高度
pWindowDC->StretchBlt(m_MaxRect.left,m_MaxRect.top,m_MaxRect.right
,m_MaxRect.bottom,&memDC,0,0,x,y,SRCCOPY); //绘制图片
LeftLine.DeleteObject();
/*****绘制最大化按钮*****/

/*****绘制关闭按钮*****/
LeftLine.LoadBitmap(IDB_BITMAP6); //加载图片资源
LeftLine.GetObject(sizeof(bitinfo),&bitinfo); //获取位图大小
memDC.SelectObject(&LeftLine); //选入位图
x = bitinfo.bmiHeader.biWidth; //获得宽度
y = bitinfo.bmiHeader.biHeight; //获得高度

```

```
pWindowDC->StretchBlt(m_CloseRect.left,m_CloseRect.top,m_CloseRect.right
,m_CloseRect.bottom,&memDC,0,0,x,y,SRCCOPY); //绘制图片
LeftLine.DeleteObject();
m_IsDrawForm = TRUE;
/*****绘制关闭按钮*****/
ReleaseDC(&memDC);
DrawFormCaption(); //绘制标题
}
```

(5) 处理窗体的 WM_NCMOUSEMOVE 消息，确定标题栏按钮的状态，代码如下：

```
void CDrawFormDlg::OnNcMouseMove(UINT nHitTest, CPoint point)
{
    CDialog::OnNcMouseMove(nHitTest, point);
    CRect tempIni,tempMin,tempMax,tempClose,ClientRect;
    CDC* pWindowDC = GetWindowDC(); //获得窗口设备上下文
    CDC memDC;
    memDC.CreateCompatibleDC(pWindowDC); //创建兼容的设备上下文
    BITMAPINFO bInfo;
    CBitmap LeftLine;
    int x,y;

    GetWindowRect(ClientRect); //获得窗口区域
    tempIni.CopyRect(CRect(m_IniRect.left+
        ClientRect.left,ClientRect.top+m_IniRect.top,m_IniRect.right+m_IniRect.left+
        ClientRect.left,m_IniRect.bottom+m_IniRect.top+ClientRect.top)); //复制初始化按钮区域
    tempMin.CopyRect(CRect(m_MinRect.left+
        ClientRect.left,ClientRect.top+m_MinRect.top,m_MinRect.right+m_MinRect.left+
        ClientRect.left,m_MinRect.bottom+m_MinRect.top+ClientRect.top)); //复制最小化按钮区域
    tempMax.CopyRect(CRect(m_MaxRect.left+
        ClientRect.left,ClientRect.top+m_MaxRect.top,m_MaxRect.right+m_MaxRect.left+
        ClientRect.left,m_MaxRect.bottom+m_MaxRect.top+ClientRect.top)); //复制最大化按钮区域
    tempClose.CopyRect(CRect(m_CloseRect.left+
        ClientRect.left,ClientRect.top+m_CloseRect.top,m_CloseRect.right+
        m_CloseRect.left+ClientRect.left,m_CloseRect.bottom+m_CloseRect.top
        +ClientRect.top)); //复制关闭按钮区域
    if (tempIni.PtInRect(point) //鼠标在初始化按钮上移动时，更改按钮显示的位图
    {
        LeftLine.LoadBitmap(IDB_BITMAP7); //加载图片资源
        LeftLine.GetObject(sizeof(bInfo),&bInfo); //获得位图信息
        x = bInfo.bmiHeader.biWidth; //获得位图宽度
        y = bInfo.bmiHeader.biHeight; //获得位图高度
        memDC.SelectObject(&LeftLine); //选入位图
        pWindowDC->StretchBlt(m_IniRect.left,m_IniRect.top,m_IniRect.right,
            m_IniRect.bottom,&memDC,0,0,x,y,SRCCOPY); //绘制位图
        m_IsDrawForm = FALSE;
        m_ButtonState = bsIni;
        LeftLine.DeleteObject();
    }
    else if (tempMin.PtInRect(point) //鼠标在最小化按钮上移动时，更改按钮显示的位图
    {
        LeftLine.LoadBitmap(IDB_BITMAP7); //加载图片资源
        LeftLine.GetObject(sizeof(bInfo),&bInfo); //获得位图信息
        x = bInfo.bmiHeader.biWidth; //获得位图宽度
        y = bInfo.bmiHeader.biHeight; //获得位图高度
        memDC.SelectObject(&LeftLine); //选入位图
        pWindowDC->StretchBlt(m_MinRect.left,m_MinRect.top,m_MinRect.right,
            m_MinRect.bottom,&memDC,0,0,x,y,SRCCOPY); //绘制位图
        m_IsDrawForm = FALSE;
        m_ButtonState = bsMin;
        LeftLine.DeleteObject();
    }
    else if (tempMax.PtInRect(point) //鼠标在最大化按钮上移动时，更改按钮显示的位图
    {
        LeftLine.LoadBitmap(IDB_BITMAP7); //加载图片资源
        LeftLine.GetObject(sizeof(bInfo),&bInfo); //获得位图信息
        x = bInfo.bmiHeader.biWidth; //获得位图宽度
        y = bInfo.bmiHeader.biHeight; //获得位图高度
        memDC.SelectObject(&LeftLine); //选入位图
        pWindowDC->StretchBlt(m_MaxRect.left,m_MaxRect.top,
            m_MaxRect.right,m_MaxRect.bottom,&memDC,0,0,x,y,SRCCOPY); //绘制位图
        m_IsDrawForm = FALSE;
        if (m_IsMax)
        {
            m_ButtonState = bsMax; //最大化
        }
        else
        {
            m_ButtonState = bsRes; //还原
        }
    }
}
```

```

    }

    LeftLine.DeleteObject();
}
else if (tempClose.PtInRect(point))           //鼠标在关闭按钮上移动时, 更改按钮显示的位图
{
    LeftLine.LoadBitmap(IDB_BITMAP7);         //加载图片资源
    LeftLine.GetObject(sizeof(bInfo), &bInfo); //获得位图信息
    x = bInfo.bmiHeader.biWidth;              //获得位图宽度
    y = bInfo.bmiHeader.biHeight;             //获得位图高度
    memDC.SelectObject(&LeftLine);            //选入位图
    pWindowDC->StretchBlt(m_CloseRect.left, m_CloseRect.top,
        m_CloseRect.right, m_CloseRect.bottom, &memDC, 0, 0, x, y, SRCCOPY); //绘制位图
    m_IsDrawForm = FALSE;
    m_ButtonState = bsClose;
    LeftLine.DeleteObject();
}
else
{
    m_ButtonState = bsNone;
    if (m_IsDrawForm == FALSE)
        DrawForm();
}
ReleaseDC(&memDC);
}

```

(6) 处理窗体的 WM_NCLBUTTONDOWN 消息, 当用户在非客户区域单击时, 根据按钮的状态执行相应操作, 代码如下:

```

void CDrawFormDlg::OnNcLButtonDown(UINT nHitTest, CPoint point)
{
    CDialog::OnNcLButtonDown(nHitTest, point);
    switch (m_ButtonState)
    {
        case bsClose:                //关闭窗口
        {
            DestroyWindow();
        }
        break;

        case bsIni:                  //还原窗口到初始大小和位置
        {
            m_IsMax = TRUE;
            MoveWindow(m_OrigonRect.left, m_OrigonRect.top, m_OrigonRect.Width(),
                m_OrigonRect.Height()); //移动窗口位置
        }
        break;

        case bsMin:                  //最小化窗口
        {
            CWnd* pDesk = GetDesktopWindow();
            CRect rect;
            pDesk->GetClientRect(rect); //获得客户区域
            SetWindowPos(0, (rect.Width()-m_OrigonRect.Width())/2,
                2, m_OrigonRect.Width(), 0, SWP_SHOWWINDOW); //移动窗口位置
        }
        break;

        case bsMax:                  //最大化窗口
        {
            m_ButtonState = bsMax;
            ShowWindow(SW_SHOWMAXIMIZED); //最大化窗口
            m_IsMax = FALSE;
        }
        break;

        case bsRes:                  //还原窗口
        {
            ShowWindow(SW_RESTORE); //还原窗口
            m_IsMax = TRUE;
        }
        break;
    }
}

```

举一反三

根据本实例, 读者可以:

- 设计个性的窗体界面。

实例 025 以时钟显示界面

本实例可以美化界面、简化操作

实例位置: 光盘\mingrisoft\01\025

实例说明

在当今紧张的生活和工作中,时间已经成为人们极为重要的财富,但是有些应用程序并不能显示时间,这就给用户带来了极大的不便。运行本实例,将根据系统时间在窗体中显示一个时钟,如图 1.31 所示。

技术要点

本实例通过 GetCurrentTime 函数获取系统时间,装入一个时钟的位图,在时钟位图中间部分使用 CreateEllipticRgn 函数创建一个椭圆形的区域,在该区域中绘制表针。

实现过程



图 1.31 以时钟显示界面窗体

- (1) 新建一个基于单文档的应用程序。
- (2) 在 ResourceView 视图中右击,选择 Import 选项,在 Import Resource 窗口中添加一个表盘位图。

- (3) 在视图类的 OnDraw 方法中绘制时钟背景,并设置定时器代码如下:

```
void CSZxscxjmView::OnDraw(CDC* pDC)
{
    CSZxscxjmDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    time = CTime::GetCurrentTime();           //获得系统时间
    if(hour>12)                               //将24小时制转换为12小时制
    {
        hour=hour-12;
    }
    sec = time.GetSecond();                   //获得秒数
    min = time.GetMinute();                   //获得分钟
    hour = time.GetHour();                     //获得小时
    CDC dc;
    CBitmap bmp;
    bmp.LoadBitmap(IDB_BITMAP1);              //IDB_BITMAP1是待显示位图的资源ID
    BITMAP bm;
    bmp.GetBitmap(&bm);                       //获得位图信息
    int nWidth = bm.bmWidth;                  //获得位图宽度
    int nHeight = bm.bmHeight;                 //获得位图高度
    dc.CreateCompatibleDC(pDC);               //创建兼容的设备上下文
    dc.SelectObject(&bmp);                     //选入位图对象
    pDC->BitBlt(18, 0, nWidth, nHeight, &dc, 0, 0, SRCCOPY); //绘制位图背景
    SetTimer(1,1,NULL);                       //设置定时器
}
```

- (4) 在定时器处理函数中绘制表针,代码如下:

```
void CSZxscxjmView::OnTimer(UINT nIDEvent)
{
    KillTimer(1);
    CDC *pDC=this->GetDC();                   //获得设备上下文
    CRect m_rect;
    this->GetClientRect(m_rect);               //获得客户区域
    CRgn rgm;
    HRGN m_hrgn;
    m_hrgn = ::CreateEllipticRgn(64,40,186,165); //创建椭圆区域
    rgm.Attach(m_hrgn);
    CBrush m_brush(1,RGB(255,255,255));       //创建画刷
    pDC->SelectClipRgn(&rgm,0);
    pDC->FillRgn(&rgm,&m_brush);               //用画刷填充区域
    int x=120;
    int y=100;
    CPen pen;                                 //声明画笔
    pen.CreatePen(PS_SOLID,1,RGB(255,0,0));  //创建实线画笔
}
```



```

pDC->SelectObject(&pen);           //将画笔选入设备上下文
pDC->MoveTo(x,y);
pDC->LineTo(x+(long)55*cos(pi/2-2*pi*sec/60.0),y-(long)55*sin(pi/2-2*pi*sec/60.0)); //绘制秒针
CPen pen1;                          //声明画笔
pen1.CreatePen(PS_SOLID,2,RGB(0,0,0)); //创建实线画笔
pDC->SelectObject(&pen1);           //将画笔选入设备上下文
pDC->MoveTo(x,y);
pDC->LineTo(x+(long)45*cos(pi/2-2*pi*min/60.0),y-(long)45*sin(pi/2-2*pi*min/60.0)); //绘制分针
pDC->MoveTo(x,y);
pDC->LineTo(x+(long)35*cos(pi/2-5*2*pi*hour/60.0),y-(long)35*sin(pi/2-5*2*pi*hour/60.0)); //绘制时针
sec = sec+1;
if(sec==60)
{
    sec=0;
    min=min+1;
    if(min==60)
    {
        min=0;
        hour=hour+1;
    }
}
SetTimer(1,1000,NULL);
CView::OnTimer(nIDEvent);
pen.DeleteObject();
pen1.DeleteObject();
}

```

举一反三

根据本实例，读者可以：

- 实现在单文档的程序中显示位图。

实例 026 窗体融合技术

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\01\026

实例说明

在 Visual C++ 6.0 的集成开发环境中，可以看到许多拖动的窗口。用户可以将其拖动到任何位置，也可以将它们停靠在窗口的一边。本实例实现了该功能，效果如图 1.32 所示。

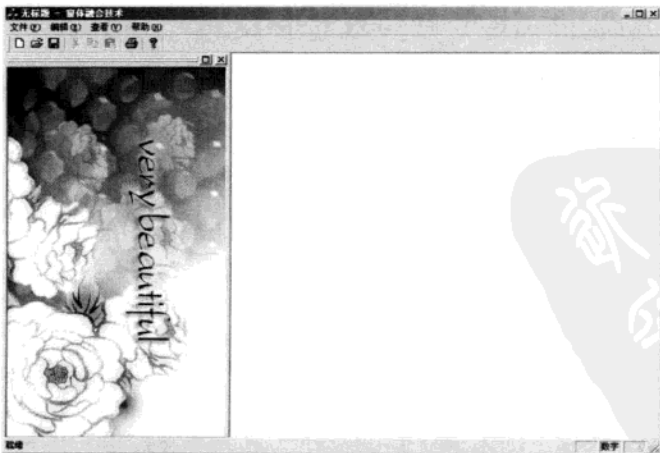


图 1.32 窗体融合技术

技术要点

在文档/视图结构的应用程序中，默认情况下，用户可以将工具栏拖动到任意位置。如果

要将对话框拖动到任意位置,就需要从控制条 CControlBar 派生一个子类,本实例为 CDockBarCtrl。在 CDockBarCtrl 中实现某个对话框的显示,并实现控制条的拖动功能。


实现对话框的显示非常容易,只要在 CDockBarCtrl 类中定义一个对话框 CDialog 指针,然后在 CDockBarCtrl 类的 Create 方法中创建对话框,并设置对话框的位置就可以了。

要实现控制条的拖动,最核心的问题是设置控制条的位置和大小,即根据当前拖动的状况(在上、下、左、右停靠,或者处于浮动状态)适当设置控制条的位置和大小。在 CControlBar 类中提供了两个虚拟方法 CalcDynamicLayout 和 CalcFixedLayout 来计算控制条的大小。在 CalcDynamicLayout 方法中只是直接调用了 CalcFixedLayout 方法,而 CalcFixedLayout 只是简单地设置控制条的大小,并没有根据实际情况进行计算。因为 CControlBar 是一个抽象类,CalcFixedLayout 方法只是进行了默认的调整。有关 CDockBarCtrl 类对 CalcDynamicLayout 和 CalcFixedLayout 方法的改写可以参考实现过程。

在控制条 CDockBarCtrl 中还应该包含一个标题栏,用于显示还原按钮、关闭按钮和两个边条。可以通过处理控制条的 WM_NCCALCSIZE 消息来设置标题栏的区域。需要注意的是标题栏的区域不是一成不变的,当控制条在上、下、左、右停靠时,标题栏的区域都是不同的。因此需要在 WM_NCCALCSIZE 消息处理函数中针对不同的情况进行设置。

```
void CDockBarCtrl::OnNcCalcsz(BOOL bCalcValidRects, NCCALCSIZE_PARAMS FAR* lpncsp)
```

```
{
    //根据每一种排列方式计算出非客户区域
    switch (m_DockBarID)
    {
        case AFX_IDW_DOCKBAR_TOP:                //控制条停靠在上
        {
            lpncsp->rgrc[0].left += m_ncTBandHeight;
            lpncsp->rgrc[0].bottom -= m_ncRBandWidth;
            lpncsp->rgrc[0].top += m_ncLBandWidth;
            lpncsp->rgrc[0].right -= m_ncBBandHeight;
            m_ncTopHeight = m_ncRBandWidth;
            break;
        }
        case AFX_IDW_DOCKBAR_LEFT:              //控制条停靠在左
        {
            lpncsp->rgrc[0].left += m_ncLBandWidth;
            lpncsp->rgrc[0].bottom -= m_ncBBandHeight;
            lpncsp->rgrc[0].top += m_ncTBandHeight;
            lpncsp->rgrc[0].right -= m_ncRBandWidth;
            m_ncLeftWidth = m_ncLBandWidth;
            break;
        }
        case AFX_IDW_DOCKBAR_RIGHT:             //控制条停靠在右
        {
            lpncsp->rgrc[0].left += m_ncRBandWidth;
            lpncsp->rgrc[0].bottom -= m_ncBBandHeight;
            lpncsp->rgrc[0].top += m_ncTBandHeight;
            lpncsp->rgrc[0].right -= m_ncLBandWidth;
            m_ncLeftWidth = m_ncLBandWidth;
            break;
        }
        case AFX_IDW_DOCKBAR_BOTTOM:           //控制条停靠在下方
        {
            lpncsp->rgrc[0].left += m_ncTBandHeight;
            lpncsp->rgrc[0].bottom -= m_ncLBandWidth;
            lpncsp->rgrc[0].top += m_ncRBandWidth;
            lpncsp->rgrc[0].right -= m_ncBBandHeight;
            m_ncTopHeight = m_ncRBandWidth;
            break;
        }
    }
}
```

 注意: CControlBar 是一个抽象类,要从该类派生一个子类,必须实现抽象方法 OnUpdateCmdUI。

实现过程

- (1) 新建一个文档/视图结构的应用程序。
- (2) 创建一个对话框类 CTools, 设置对话框

属性如图 1.33 所示。

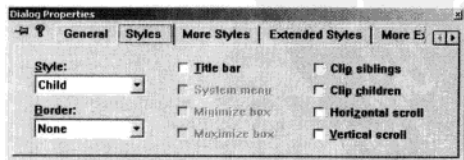


图 1.33 对话框属性设置

- (3) 从 CControlBar 派生一个子类 CDockBarCtrl, 实现抽象方法 OnUpdateCmdUI, 代码如下:

```
void CDockBarCtrl::OnUpdateCmdUI(CFrameWnd* pTarget, BOOL bDisableIfNoHndler)
{
    CWnd::UpdateDialogControls(pTarget, bDisableIfNoHndler);
}
```

(4) 在 CDockBarCtrl 类中定义如下成员变量:

```
CDialog *m_pDlg;           //关联的对话框
CBrush m_bkBrush;          //背景画刷
CString m_Caption;         //窗口标题
CRect m_FloatRect;         //浮动时的窗口区域
CRect m_VerRect;           //拖入时, 在左、右显示时的区域
CRect m_HorRect;           //拖入时, 在上、下显示时的区域
//确定控制条的非客户区域部分
int m_ncTBandHeight;        //非客户区域上边条高度
int m_ncRBandWidth;         //右边条的宽度
int m_ncLBandWidth;         //确定左边条的宽度
int m_ncBBandHeight;        //确定底边条的宽度
//定义标题栏按钮的显示区域
CRect m_closeRC;           //关闭按钮的显示区域
CRect m_maxRC;             //最大化按钮的显示区域
int m_ncLeftWidth;          //记录垂直显示时的非客户区域左边条宽度
int m_ncTopHeight;          //记录水平显示时的非客户区域上边条的宽度
COLORREF m_TopBandColor, m_BottomBandColor; //标题线条的颜色
BOOL m_IsTraking;           //是否处于调整控制条大小状态
BOOL m_IsInRect;            //判断控制条是否处于停靠状态
CPoint m_OldPos;            //鼠标按下时原始点位置
CRect m_Bandrc;             //拖动的边条区域
CRect m_temprc;             //临时的区域
int m_TopInterval, m_LeftInterval; //控制条相对父窗口的位置
int m_HitTest;              //当控制条处于浮动状态时, 限制用户改变控制条的最小区域
CSize m_Min;                //当控制条处于浮动状态时, 限制用户改变控制条的最小区域
```

(5) 在 CDockBarCtrl 类中改写 Create 方法, 创建控制条窗口和对话框窗口, 代码如下:

```
BOOL CDockBarCtrl::Create(CWnd* pParentWnd, CDialog* pDialog, UINT nID, DWORD dwStyle,
    CCreateContext* pContext)
{
    ASSERT_VALID(pParentWnd);
    ASSERT(!((dwStyle & CBRS_SIZE_DYNAMIC) && (dwStyle & CBRS_SIZE_FIXED)));
    m_dwStyle = dwStyle & CBRS_ALL;
    //注册窗口类
    CString classname;
    classname = AfxRegisterWndClass(CS_DBLCLKS, LoadCursor(NULL, IDC_ARROW), m_bkBrush, 0);
    //创建窗口类
    CWnd::Create(classname, m_Caption, dwStyle, CRect(0, 0, 0, 0), pParentWnd, 0);
    m_pDlg = pDialog;
    m_pDlg->Create(nID, this);           //创建对话框
    m_pDlg->GetWindowRect(m_FloatRect); //获得对话框窗口区域
    m_HorRect.CopyRect(m_FloatRect);   //复制对话框区域
    m_VerRect.CopyRect(m_FloatRect);   //复制对话框区域
    return true;
}
```

(6) 在 CDockBarCtrl 类中改写 CalcFixedLayout 方法, 计算控制条停靠时的大小, 代码如下:

```
CSize CDockBarCtrl::CalcFixedLayout(BOOL bStretch, BOOL bHorz)
{
    int dockwidth, dockheight;
    CRect rc_clientrc;
    m_pDockSite->GetClientRect(rc_clientrc); //获得客户区域
    //如果当前处于浮动状态
    if (IsFloating())
        return CSize(m_FloatRect.Width(), m_FloatRect.Height()); //返回对话框的宽度和高度
    else //处于固定状态
    {
        if (bHorz) //水平方向显示
        {
            m_pDockSite->GetControlBar(AFX_IDW_DOCKBAR_TOP)->GetWindowRect(rc);
            dockwidth = bStretch ? 1200:rc.Width()+4;
            return CSize(dockwidth, m_HorRect.Height());
        }
        else //垂直方向显示
        {
            m_pDockSite->GetControlBar(AFX_IDW_DOCKBAR_LEFT)->GetWindowRect(rc);
            dockheight = bStretch ? 1200:rc.Height()+4;
            return CSize(m_VerRect.Width(), dockheight);
        }
    }
}
```

(7) 在 CDockBarCtrl 类中改写 CalcDynamicLayout 方法, 根据控制条的不同状态设置控制

条大小,代码如下:

```
CSize CDockBarCtrl::CalcDynamicLayout(int nLength, DWORD nMode)
{
    if (IsFloating())
    {
        //修改CMiniDockFrameWnd的风格
        //允许同时调整窗口的宽度和高度
        CFrameWnd* pDockFrame = GetDockingFrame();
        pDockFrame->ModifyStyle(MFS_4THICKFRAME,0);
    }
    if (nMode&(LM_HORZDOCK|LM_VERTDOCK))
        return CControlBar::CalcDynamicLayout(nLength,nMode);
    if (nMode&LM_COMMIT)
        return CSize(nLength,m_FloatRect.Height());
    if (nMode & LM_MRWIDTH)
        return CSize(m_FloatRect.Width(),m_FloatRect.Height());
    CRect rect;
    GetParent()->GetParent()->GetWindowRect(rect);           //获取CMiniDockFrameWnd的窗口区域
    int CaptionHeight = GetSystemMetrics(SM_CYCAPTION);       //获取CMiniDockFrameWnd的标题栏高度
    if (IsFloating())
    {
        CPoint pt;
        GetCursorPos(&pt);                                     //获取鼠标当前位置
        m_FloatRect.left = 0;
        m_FloatRect.top = 0;
        //判断用户在窗体的哪个角开始调整大小,实际上m_pDockContext->m_nHitTest的值
        //是在CMiniDockFrameWnd的OnNcLButtonDown方法中调用m_pDockContext的StartResize方法设置的
        switch (m_pDockContext->m_nHitTest)
        {
            case HTTOPLEFT:                                     //鼠标在左上角
            {
                m_FloatRect.left = 0;
                m_FloatRect.top = 0;
                m_FloatRect.right = (m_Min.cx > rect.right - pt.x)?m_Min.cx: (rect.right - pt.x); //调整宽度
                int temp = rect.bottom - CaptionHeight - pt.y - 1;
                m_FloatRect.bottom = (m_Min.cy > temp)? m_Min.cy : temp; //调整高度
                //调整CMiniDockFrameWnd的左上角坐标为当前移动的鼠标坐标
                m_pDockContext->m_rectFrameDragHorz.top = pt.y;
                m_pDockContext->m_rectFrameDragHorz.left = pt.x;
                return CSize(m_FloatRect.Width(),m_FloatRect.Height());
            }
            case HTTOPRIGHT:                                    //鼠标在右上角
            {
                m_FloatRect.left = 0;
                m_FloatRect.top = 0;
                m_FloatRect.right = (m_Min.cx > pt.x-rect.left)?m_Min.cx: pt.x-rect.left; //调整宽度
                int temp = rect.bottom - CaptionHeight - pt.y - 1;
                m_FloatRect.bottom = (m_Min.cy > temp)? m_Min.cy : temp; //调整高度
                m_pDockContext->m_rectFrameDragHorz.top = pt.y;
                return CSize(m_FloatRect.Width(),m_FloatRect.Height());
            }
            case HTBOTTOMLEFT:                                   //鼠标在左下角
            {
                m_FloatRect.right = max(m_Min.cx, rect.right-pt.x); //调整宽度
                m_FloatRect.bottom = max(m_Min.cy,pt.y-rect.top-CaptionHeight); //调整高度
                m_pDockContext->m_rectFrameDragHorz.left = max(m_Min.cx,pt.x);
                return CSize(m_FloatRect.Width(),m_FloatRect.Height());
            }
            case HTBOTTOMRIGHT:                                  //鼠标在右下角
            {
                m_FloatRect.right = max(m_Min.cx, pt.x-rect.left); //调整宽度
                m_FloatRect.bottom = max(m_Min.cy,pt.y-rect.top-CaptionHeight); //调整高度
                m_pDockContext->m_rectFrameDragHorz.right = pt.x;
                return CSize(m_FloatRect.Width(),m_FloatRect.Height());
            }
        }
    }
    if (nMode&LM_LENGTHY)                                     //调整控制条的高度
    {
        m_FloatRect.top=0;
        m_FloatRect.bottom= max(m_Min.cy, nLength);
        return CSize(m_FloatRect.Width(),max(m_Min.cy, nLength));
    }
    else if (nMode&LM_HORZ)                                    //调整控制条的宽度
    {
        m_FloatRect.top = 0;
        m_FloatRect.left = 0;
```



```
m_FloatRect.right = max(m_Min.cx, nLength); //调整宽度
m_FloatRect.bottom = max(m_Min.cy, rect.Height() - CaptionHeight); //调整高度
return CSize(max(nLength, m_Min.cx), m_FloatRect.Height());
}
```

举一反三

根据本实例，读者可以：

- 设计动画效果的控制条。

实例 027

限制对话框最大时的窗口大小

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\01\027

实例说明

在设计程序界面时，有时需要限制对话框的大小。例如，将对话框限制在某一范围内。本实例实现了该功能，效果如图 1.34 所示。



图 1.34 限制对话框最大时的窗口大小

技术要点

当对话框的大小和位置发生改变时，会接收到 WM_GETMINMAXINFO 消息。用户只要在该消息处理函数中设置对话框的大小就可以了。WM_GETMINMAXINFO 消息处理函数语法如下：

```
afx_msg void OnGetMinMaxInfo( MINMAXINFO FAR* lpMMI );
```

参数说明：

- lpMMI：是 MINMAXINFO 结构指针，该结构记录着对话框最大化、最小化时的大小，用于限制对话框大小。其中，ptMaxSize 成员用于设置对话框最大化时的高度和宽度，ptMaxPosition 成员标识对话框最大化时的位置。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加图片控件。
- (3) 处理对话框的 WM_GETMINMAXINFO 消息，限制对话框的大小，代码如下：

```
void CLimitSizeDlg::OnGetMinMaxInfo(MINMAXINFO FAR* lpMMI)
{
    lpMMI->ptMaxSize.x = 800; //设置对话框最大化时的宽度
    lpMMI->ptMaxSize.y = 600; //设置对话框最大化时的高度
    lpMMI->ptMaxPosition.x = 50; //设置对话框最大化时左边的位置
    lpMMI->ptMaxPosition.y = 50; //设置对话框最大化时上方的位置
}
```

```
CDialog::OnGetMinMaxInfo(lpMMI);
```

举一反三

根据本实例，读者可以：

- 实现限制对话框最小化时的大小。

实例 028 分割视图窗口

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\01\028

实例说明

在文档/视图结构中，视图用来显示文档中的数据，也可以对视图进行分割，从而使视图显示不同的信息。本实例实现了分割视图窗口的功能，效果如图 1.35 所示。

技术要点

通过 CSplitterWnd 类可以实现视图分割，当改变分割窗口的大小时，窗口的客户区将自动重新绘制。首先调用 CreateStatic 方法创建静态分割的窗体，然后调用 CreateView 方法创建子视图。CreateStatic 方法语法如下：

```
BOOL CreateStatic( CWnd* pParentWnd, int nRows, int nCols, DWORD dwStyle = WS_CHILD | WS_VISIBLE, UINT nID = AFX_IDW_PANE_FIRST );
```

参数说明：

- pParentWnd：分割窗体的父窗体对象，一般为框架窗体对象。
- nRows：分割窗体的行数，该值不能超过 16。
- nCols：分割窗体的列数，该值不能超过 16。
- dwStyle：设置分割后窗体的样式。
- nID：设置被创建对象所使用的资源 ID 值。

CreateView 方法语法如下：

```
virtual BOOL CreateView( int row, int col, CRuntimeClass* pViewClass, SIZE sizeInit, CCreateContext* pContext );
```

参数说明：

- row：子视图所在行。
- col：子视图所在列。
- pViewClass：新视图的 CRuntimeClass 对象。
- sizeInit：指定新视图的初始大小。
- pContext：指向 CCreateContext 结构的指针。

实现过程

- (1) 新建一个基于单文档的应用程序。
- (2) 以 CTreeView 类为基类派生一个 CMyTreeView 类，以 CListView 类为基类派生一个 CMyListView 类。

(3) 主要程序代码如下：

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
```

```
{
    split.CreateStatic(this,1,2);                                //创建静态分割窗口
    split.CreateView(0,1,RUNTIME_CLASS(CMyListView),CSize(150,100),pContext); //创建列表子视图
    split.CreateView(0,0,RUNTIME_CLASS(CMyTreeView),CSize(200,100),pContext);  //创建树状子视图
}
```

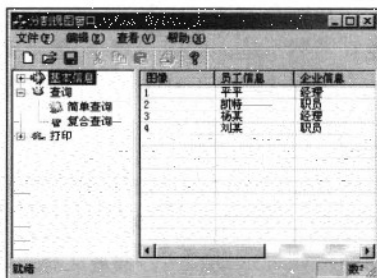


图 1.35 分割视图窗口

```
return CFrameWnd::OnCreateClient(lpcs, pContext);
```

举一反三

根据本实例，读者可以：

- 实现视图的自由分割。

实例 029 Animate 动画显示窗体

本实例是一个提高效率的程序

实例位置：光盘\mingr\soft\01\029

实例说明

通常在窗体显示时并不会有任何的动画效果，那么是不是在窗体显示时就不能存在动画效果呢？当然不是，利用 API 函数是完全可以实现窗体的动画显示的。本例就实现了窗体显示时的动画效果，效果如图 1.36 所示。

技术要点

要实现动画显示窗体效果需要使用 `AnimateWindow` 函数，并设置 `0x00000010` 风格，由于该函数并没有被封装，所以需要手动导入 `User32` 动态库，并定义 `0x00000010` 风格为 `AW_CENTER`。该函数的使用可以通过在窗口的创建或销毁过程中运用，可开启和关闭程序时达到所希望的动画窗口效果。`AnimateWindow` 函数所提供的动画效果十分丰富，可以在自己的程序中选择各种不同的动画效果，增强程序的趣味性。`AnimateWindow` 函数语法如下：

```
BOOL AnimateWindow (HWND hWnd, DWORD dwTime, DWORD dwFlags)
```

参数说明：

- `hWnd`：指定产生动画的窗口的句柄。
- `dwTime`：指明动画持续的时间（以微秒计），完成一个动画的标准时间为 200 微秒。
- `dwFlags`：指定动画类型。这个参数可以是一个或多个下列标志的组合。标志描述如下。
 - `AW_SLIDE`：使用滑动类型。缺省则为滚动动画类型。当使用 `AW_CENTER` 标志时，这个标志就被忽略。
 - `AW_ACTIVATE`：激活窗口。在使用了 `AW_HIDE` 标志后不能使用这个标志。
 - `AW_BLEND`：实现淡出效果。只有当 `hWnd` 为顶层窗口的时候才可以使用此标志。
 - `AW_HIDE`：隐藏窗口，缺省则显示窗口。
 - `AW_CENTER`：若使用了 `AW_HIDE` 标志，则使窗口向内重叠，即收缩窗口；若未使用 `AW_HIDE` 标志，则使窗口向外扩展，即展开窗口。
 - `AW_HOR_POSITIVE`：自左向右显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 `AW_CENTER` 标志时，该标志将被忽略。
 - `AW_VER_POSITIVE`：自顶向下显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 `AW_CENTER` 标志时，该标志将被忽略。
 - `AW_VER_NEGATIVE`：自下向上显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 `AW_CENTER` 标志时，该标志将被忽略。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。

实现过程

- (1) 创建一个基于对话框的应用程序。

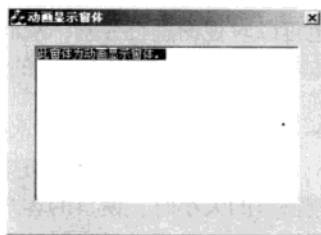


图 1.36 对话框全屏显示

- (2) 向对话框中添加一个文本编辑控件。
 (3) 在窗体初始化方法 OnInitDialog 中添加窗体动画效果, 代码如下:

```
m_edit = "此窗体为动画显示窗体。";
this->UpdateData(false);
// TODO: Add extra initialization here
CenterWindow(); //创建窗体
DWORD dwStyle=AW_CENTER; //居中
HINSTANCE hInst=LoadLibrary("User32.DLL"); //载入动态库
typedef BOOL (WINAPI MYFUNC)(HWND,DWORD,DWORD); //定义函数类型
MYFUNC* AnimateWindow; //定义函数指针
AnimateWindow=(MYFUNC *)::GetProcAddress(hInst,"AnimateWindow"); //获取函数地址
AnimateWindow(this->m_hWnd,1000,dwStyle); //设置窗体动画
FreeLibrary(hInst); //释放动态库
```

- (4) 在窗体的关闭事件中添加窗体动画代码, 代码如下:

```
void CDonghuaDlg::OnClose()
{
    DWORD dwStyle=AW_CENTER; //居中动画
    HINSTANCE hInst=LoadLibrary("User32.DLL"); //载入动态库
    typedef BOOL (WINAPI MYFUNC)(HWND,DWORD,DWORD); //定义函数类型
    MYFUNC* AnimateWindow; //定义函数指针
    AnimateWindow=(MYFUNC *)::GetProcAddress(hInst,"AnimateWindow"); //获取函数地址
    AnimateWindow(this->GetSafeHwnd(),700,AW_HIDE|dwStyle); //设置窗体动画
    FreeLibrary(hInst); //释放动态库
    CDialog::OnClose();
}
```

举一反三

根据本实例, 读者可以:

- 用不同的动画显示窗体。

1.7 多媒体宣传光盘应用实例

多媒体宣传光盘以其自身的特点和优势被广泛的应用在传媒领域。本节通过几个实例介绍多媒体宣传光盘的制作。

实例 030 多媒体宣传光盘主界面

本实例是一个提高效率的程序

实例位置: 光盘\mingrisoft\01\030

实例说明

随着影视行业的不断发展, 优秀的电影层出不穷。为了宣传影片, 制片公司采取了多种手段。多媒体宣传光盘不但可以让用户了解影片的简介, 还可以提供精彩的影视片段, 是其他宣传手段无法取代的。本实例实现了多媒体宣传光盘的主界面, 效果如图 1.37 所示。

技术要点

本实例界面的背景由一幅图片构成, 导航按钮则使用静态文本控件实现。在使用静态文本控件时, 默认情况下, 该控件不会发送通知消息到父窗口, 因此不会响应鼠标的单击事件。在静态文本控件的属性窗口中选中 Notify 复选框, 然后修改静态文本控件的 ID, 在类向导窗口中处理 BN_CLICKED 消息, 如图 1.38 所示。



图 1.37 多媒体宣传光盘主界面

实现过程

- (1) 新建一个基于对话框的应用程序。



图 1.38 静态文本的单击事件

(2) 在对话框中添加图片控件、静态文本控件。

(3) 通过资源视图导入一幅位图作为背景图片，同时会生成一个位图 ID，默认为 IDB_BITMAP1。在图片控件的属性窗口中将 Type 属性修改为 Bitmap，将 Image 属性修改为 IDB_BITMAP1，如图 1.39 所示。

(4) 修改静态文本控件的属性，如图 1.40 所示。

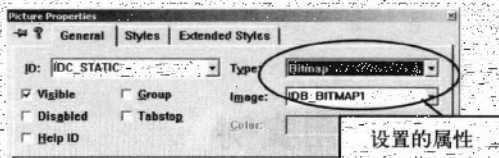


图 1.39 Picture 控件属性设置

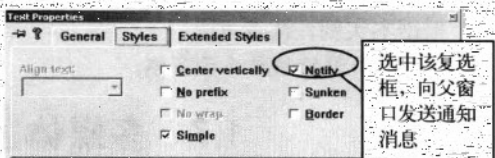


图 1.40 Static 控件属性设置

(5) 处理静态文本控件的单击事件，代码如下：

```
void CMediaDlg::OnQuit()
{
    OnCancel();
}
```

举一反三

根据本实例，读者可以：

- 设计企业多媒体宣传光盘；
- 开发多媒体动画展示光盘。

实例 031

自动运行的多媒体宣传光盘

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\01\031

实例说明

在各种应用软件的安装盘中都具有自动运行的功能，将安装盘放入光驱中，会自动运行，出现安装画面。本实例是一个光盘自动运行的程序，效果如图 1.41 所示。

技术要点

其实，实现光盘的自动运行非常简单。当用户打开自动运行的光盘时，会发现光盘中有几个特殊的文件，分别为



图 1.41 自动运行的多媒体宣传光盘

“autorun.exe”、“run.ico”和“autorun.inf”，其中“autorun.exe”是光盘自动播放时执行的可执行文件，“run.ico”是光盘的图标，“autorun.inf”是一个INI文件。只要光盘中具有这3个文件，就会自动运行。

本实例中涉及操作INI文件，Visual C++提供了WritePrivateProfileString函数，用于创建并写入INI文件数据，该函数语法如下：

```
BOOL WritePrivateProfileString(LPCTSTR lpAppName, LPCTSTR lpKeyName, LPCTSTR lpString, LPCTSTR lpFileName);
```

参数说明：

- lpAppName：用于指定INI文件中的段名。
- lpKeyName：用于指定INI文件中的键名。
- lpString：用于指定键值。
- lpFileName：用于指定INI文件名称。
- Visual C++相应地提供了GetPrivateProfileString函数，用于读取INI文件中的数据，该函数语法如下：

```
DWORD GetPrivateProfileString(LPCTSTR lpAppName, LPCTSTR lpKeyName, LPCTSTR lpDefault, LPTSTR lpReturnedString, DWORD nSize, LPCTSTR lpFileName);
```

参数说明：

- lpAppName：标识INI文件中的段名。
- lpKeyName：标识INI文件中的键名。
- lpDefault：标识默认值。
- lpReturnedString：用于接收读取的数据。
- nSize：标识lpReturnedString的大小。
- lpFileName：标识读取的INI文件名称。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加Picture控件。
- (3) 在对话框初始化时创建INI文件，代码如下：

```
CString appname = AfxGetAppName();
appname.Insert(0, "\\");
appname.Insert(appname.GetLength(), ".INI");
WritePrivateProfileString("autorun", "open", "AUTORUN.EXE", appname);
WritePrivateProfileString("autorun", "ICON", "run.ico", appname);
```

举一反三

根据本实例，读者可以：

- 利用INI文件动态连接数据库。

1.8 多媒体触摸屏程序应用实例

随着多媒体技术的不断发展，触摸屏程序应用越来越广泛。本节主要介绍触摸屏程序的设计。

实例 032

采购中心多媒体触摸屏程序

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\01\032

实例说明

触摸屏软件是新兴的一种多媒体应用软件，与传统的键盘和鼠标标准输入方式相比，触摸

屏输入更直观、更方便。它是政府和企事业单位为百姓、客户服务的一种非常好的手段。本实例介绍政府采购中心多媒体触摸屏程序的设计,效果如图 1.42 所示。

技术要点

本实例界面由图片和 AVI 动画构成,因此核心问题是对 AVI 文件的播放。在 MFC 类库中提供了 CAnimateCtrl 类,用于播放 AVI 动画,该类的主要方法如下所示。

(1) Open 方法。该方法用于打开一个 AVI 文件。在播放 AVI 动画之前,首先应调用该方法打开文件。Open 方法语法如下:

```
BOOL Open(LPCTSTR lpszFileName);
BOOL Open(UINT nID);
```

参数说明:

- lpszFileName: 标识 AVI 文件名称。
- nID: 标识 AVI 资源 ID。

(2) Play 方法。该方法用于播放 AVI 文件,语法如下:

```
BOOL Play(UINT nFrom, UINT nTo, UINT nRep);
```

参数说明:


- nFrom: 标识播放帧的起始位置。
- nTo: 标识播放帧的终止位置。
- nRep: 标识循环次数,如果为-1,表示始终循环播放。

(3) Stop 方法。该方法用于停止 AVI 文件的播放,语法如下:

```
BOOL Stop();
```

(4) Close 方法。该方法用于关闭 AVI 文件,语法如下:

```
BOOL Close();
```

 注意: CAnimateCtrl 类只能播放无声音的 AVI 文件,如果播放有声音的 AVI 文件,需要使用微软公司 ActiveX 控件 CAnimation。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加 Picture 控件、Animate 控件。
- (3) 通过类向导为 Animate 控件命名,如图 1.43 所示。

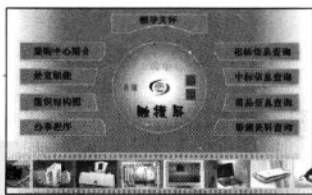


图 1.42 采购中心多媒体触摸屏程序

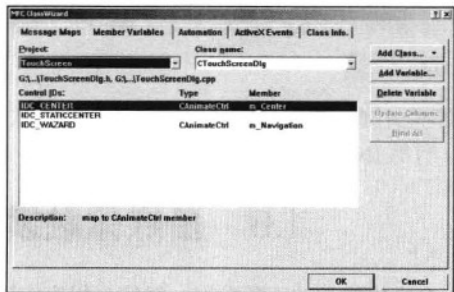


图 1.43 类向导窗口

(4) 在对话框的 OnInitDialog 方法中播放 AVI 动画,代码如下:

```
m_Navigation.Open("gd.avi");
m_Navigation.Play(0,-1,-1);
m_Center.Open("中间标.avi");
m_Center.Play(0,-1,-1);
```

举一反三

根据本实例,读者可以:

● 开发触摸屏地理定位程序。

实例 033

为触摸屏程序添加虚拟键盘

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\01\033

实例说明

触摸屏程序最大的缺点就是没有键盘输入。如果进行动态数据查找，会带来一定的麻烦。本实例通过添加一个虚拟键盘实现触摸屏程序的动态数据查找，效果如图 1.44 所示。

技术要点

要实现一个虚拟键盘可以在界面上放置一些按钮，用于数据录入。当用户单击某个按钮时，编辑框中会显示相应的数据。程序中需要处理按钮的单击事件，将当前按钮的文本显示在编辑框中。由于虚拟键盘需要多个按钮，如果为每个按钮一一处理单击事件，会很繁琐。如果编写一个通用的消息处理函数，由每个按钮在触发单击事件时调用，就可以简化代码了。

为了截获按钮单击事件，需要改写对话框的 `PreTranslateMessage` 方法，在该方法中判断当前是否触发了 `WM_LBUTTONDOWN`，如果是则调用自定义的方法。详细代码可以参考实现过程。

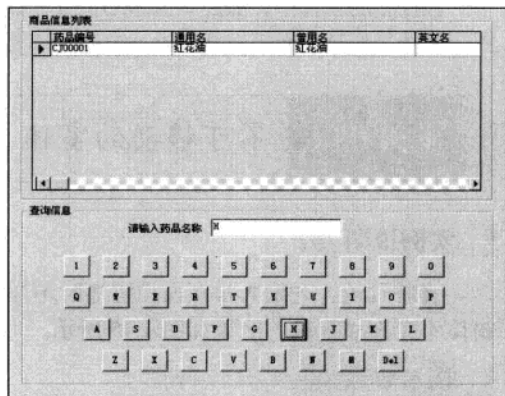


图 1.44 为触摸屏程序添加虚拟键盘

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加 `DataGrid`、编辑框、静态文本、按钮和群组控件。
- (3) 在对话框类中添加 `OnBtnClick` 方法，处理按钮的单击事件，代码如下：

```
void CVirtualKeyDlg::OnBtnClick(HWND hButton)
{
    CButton* pButton = (CButton*)CButton::FromHandle(hButton);
    if (pButton)
    {
        CString str;
        pButton->GetWindowText(str);           //获得按钮显示文本
        CString text;
        m_Data.GetWindowText(text);           //获得编辑框显示文本
        if (str != "Del")                      //如果不是Del按钮
        {
            text.Insert(text.GetLength(), str); //将按钮文本插入到编辑框文本后
        }
        else                                  //否则
        {
            text = text.Left(text.GetLength()-1); //编辑框去除最右侧字符
        }
        m_Data.SetWindowText(text);           //设置编辑框显示文本
    }
}
```

- (4) 改写对话框类的 `PreTranslateMessage` 方法，截获鼠标的单击事件，代码如下：

```
BOOL CVirtualKeyDlg::PreTranslateMessage(MSG* pMsg)
{
    if (pMsg->message == WM_LBUTTONDOWN) //获得鼠标左键抬起事件
    {
        OnBtnClick(pMsg->hwnd);          //处理当前按下按钮的单击事件
    }
}
```



```
return CDialog::PreTranslateMessage(pMsg);
```

举一反三

根据本实例,读者可以:

- 设计虚拟键盘密码验证程序。

1.9 窗体位置应用实例

在许多的软件中,都会对窗体的位置和移动进行限定,同时如何以动画方式显示窗体、如何设置窗体始终在最上面等都是在设计窗体时需要解决的问题。本节将介绍与窗体位置有关的技术。

实例 034 不可移动的窗体

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\01\034

实例说明

有时候,开发人员在设计程序时不想让用户在使用中移动窗体。运行本实例,可以发现窗体不允许被拖动,结果如图1.45所示。

技术要点

要实现不允许拖动窗体的功能,只需要通过主窗体的虚函数 `PreTranslateMessage` 截获鼠标按下时的消息,将单击标题栏的消息修改成单击非标题栏区域的消息即可。

实现过程

(1) 新建一个基于对话框的应用程序,将窗体标题改为“不可移动的窗体”。

(2) 主要程序代码如下:

```
BOOL CBKydctDlg::PreTranslateMessage(MSG* pMsg)
{
    if(pMsg->message==WM_NCLBUTTONDOWN)    //截获鼠标左键在标题栏按下消息
    {
        pMsg->message = WM_LBUTTONDOWN;    //修改为鼠标左键在非标题栏按下的消息
    }
    return CDialog::PreTranslateMessage(pMsg);
}
```



图 1.45 不可移动的窗体

举一反三

根据本实例,读者可以:

- 利用鼠标在非标题栏区域拖动窗体。

实例 035 始终在最上面的窗体

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\01\035

实例说明

在使用软件的过程中,有时会因为打开其他软件而将正在操作的软件置于其后,为操作带来了不便。

在程序运行后,无论用户打开多少个窗体;本程序的窗体始终在最上面,结果如图1.46所示。

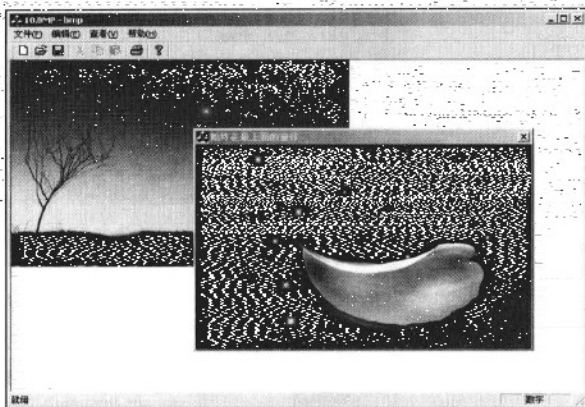


图1.46 始终在最上面的窗体

技术要点

要实现将自己的程序永远置前,可以使用 API 函数 SetWindowPos, 该函数可以为窗口指定一个新位置和状态。SetWindowPos 函数的原型如下:

```
BOOL SetWindowPos (HWND hWnd,HWND hWndInsertAfter,int X,int Y,int cx,int cy,UINT flags);
```

参数说明:

- hWnd: 窗口句柄。
- hWndInsertAfter: 在 z 轴顺序中的标识窗口的句柄, 该句柄将确定窗口出现在 z 轴中的顺序。
- X: 以客户坐标指定窗口新位置的左边界。
- Y: 以客户坐标指定窗口新位置的顶边界。
- cx: 以像素指定窗口的宽度。
- cy: 以像素指定窗口的高度。
- flags: 窗口尺寸和定位的标志。

如果将窗口置前, 可以将该函数中的 hWndInsertAfter 参数值设置为 HWND_TOPMOST。

实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“始终在最上面的窗体”。
- (2) 主要程序代码如下:

```
::SetWindowPos(AfxGetMainWnd()->m_hWnd,HWND_TOPMOST,10,10,450,300,SWP_NOMOVE);
```

举一反三

根据本实例, 读者可以:

- 设置窗口置后。

实例 036 如 QQ 般隐藏的窗体

本实例是一个人性化的程序

实例位置: 光盘\mingrsoft\01\036

实例说明

如果把一些较小的窗体做成像 QQ 般隐藏的窗体将会更加吸引用户。要实现如 QQ 般隐藏的窗体需要在定时器中判断鼠标和窗体的位置, 然后使用 MoveWindow 方法移动窗体, 从而实

现是隐藏还是显示,结果如图 1.47 所示。

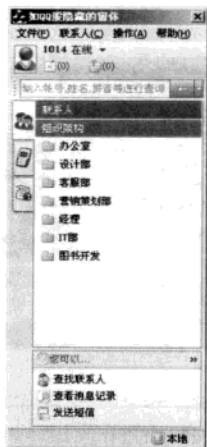


图 1.47 如 QQ 般隐藏的窗体

技术要点

窗体的隐藏主要是通过通过在定时器中获得鼠标和窗体的位置,然后调用 MoveWindow 方法实现的。该方法的语法如下:

```
BOOL MoveWindow(int X,int Y,int nWidth,int nHeight,BOOL bRepaint);
```

参数说明:

- X、Y: 窗口新位置的左上角坐标。
- nWidth: 窗口的宽度。
- nHeight: 窗口的高度。
- bRepaint: 设置窗口是否重画。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体的 OnTimer 方法中实现窗体的隐藏,代码如下:

```
void CQQHideDlg::OnTimer(UINT nIDEvent)
{
    CRect rc;
    CRect rect;
    GetWindowRect(&rect);//窗体大小
    rc.CopyRect(&rect);//复制矩形区
    CPoint pOint;
    GetCursorPos(&pOint);//获取鼠标位置
    if(rect.top < 0 && PtInRect(rect,pOint))//显示窗体
    {
        rect.top = 0;
        MoveWindow(rect.left,rect.top,rc.Width(),rc.Height());//移动窗体
    }
    else if(rect.top > -3 && rect.top < 3 && !PtInRect(rect,pOint))//隐藏窗体
    {
        rect.top = 3-rect.Height();
        MoveWindow(rect.left,rect.top,rc.Width(),rc.Height());
    }
    CDialog::OnTimer(nIDEvent);
}
```

举一反三

根据本实例,读者可以:

- 实现窗体在固定位置时隐藏，其他位置时显示。

实例 037 磁性窗体

本实例是一个人性化的程序

实例位置：光盘\mingrisoft\01\037

实例说明

用户在使用一些播放器的软件时，会发现很多的播放器都是由几个窗体组合而成的，这些窗体可以连在一起移动，也可以分开单独移动，增加了软件应用的灵活性。运行程序，调整窗体位置，如图 1.48 所示。

技术要点

实现磁性窗体功能时主要用到了 MapWindowPoints 函数和 MoveWindow 函数设置并移动窗体位置，下面对本实例中用到的关键技术进行详细讲解。

MapWindowPoints 函数：于将某个窗口的区域坐标转换为另一个窗口的区域坐标，语法如下：

```
void MapWindowPoints( CWnd* pwndTo, LPRECT lpRect ) const;
```

参数说明：

- pwndTo：表示转换后的区域坐标窗口。
- lpRect：表示待转换的区域对象。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向工程中插入两个 BMP 位图资源，向对话框中添加一个图片控件，设置其 Type 属性为 Bitmap，设置其 Image 属性为 IDB_BITMAP1。
- (3) 创建一个新的对话框资源，修改其 ID 为 IDD_CHILD_DIALOG，将其窗体标题改为“均衡器”，并设置对话框显示的字体信息。
- (4) 通过类向导为新建的对话框资源关联一个对话框类 CChildDlg。
- (5) 向新建的对话框中添加一个图片控件，设置其 Type 属性为-Bitmap，设置其 Image 属性为 IDB_BITMAP2。
- (6) 主要程序代码如下：

```
void CMagnetismDlg::OnMove(int x, int y)
{
    CDialog::OnMove(x, y);
    if(m_IsCreate == TRUE)                                //已创建
    {
        CRect pRect, cRect;                                //声明区域对象
        GetWindowRect(pRect);                              //获得主窗体区域
        MapWindowPoints(this, pRect);                      //转换窗体区域坐标
        m_Dlg->GetWindowRect(cRect);                      //获得均衡器窗体区域
        //如果移动播放器窗体距离均衡器窗体不到20像素则移动播放器窗体，使两个窗体相连
        if(pRect.left-cRect.right<20 && pRect.left-cRect.right>0 && (
            pRect.top>cRect.top-m_Height && pRect.bottom<cRect.bottom+m_Height))
            pRect.left = cRect.right;                      //设置播放器左边与均衡器右边相连
        else if(cRect.top-pRect.bottom<20 && cRect.left-pRect.right>0 && (
            pRect.top>cRect.top-m_Height && pRect.bottom<cRect.bottom+m_Height))
            pRect.left = cRect.left - m_Width;            //设置播放器右边与均衡器左边相连
        else if(cRect.top-pRect.bottom<20 && cRect.top-pRect.bottom>0 && (
            pRect.left>cRect.left-m_Width && pRect.right<cRect.right+m_Width))
            pRect.top = cRect.top - m_Height;             //设置播放器下边与均衡器上边相连
        else if(pRect.top-cRect.bottom<20 && pRect.top-cRect.bottom>0 && (
            pRect.left>cRect.left-m_Width && pRect.right<cRect.right+m_Width))
```



图 1.48 磁性窗体




```

pRect.top = cRect.bottom;           //设置播放器上边与均衡器下边相连
MoveWindow(pRect.left,pRect.top,m_Width,m_Height); //移动播放器窗体
if(m_Berth)                          //如果两个窗体相连
{
    m_Dlg->MoveWindow(cRect.left+(pRect.left-m_Point.x),
        cRect.top+(pRect.top-m_Point.y),cRect.Width(),cRect.Height()); //移动均衡器窗体
}
m_Point.x = pRect.left;             //设置左上角横坐标
m_Point.y = pRect.top;             //设置左上角纵坐标
}
}

```

举一反三

根据本实例，读者可以：

- 自由组合的磁性窗体。

1.10 窗体标题栏应用实例

为了使程序更加人性化，可以对窗体的标题栏进行设计，本节的几个实例设计的标题栏使界面看起来更美观，并具有提醒功能。

实例 038 闪烁的窗体标题栏

本实例是一个具有提醒功能的程序

实例位置：光盘\mingrisoft\01\038

实例说明

当用户运行多个程序时，为了突出显示自己开发的程序，可以设置程序窗体的标题栏闪烁，以提醒用户，使用户更多地关注该程序。

运行程序，窗体的标题栏将不停地闪烁，直到关闭程序为止，如图 1.49 所示。

技术要点

要实现窗体标题栏的闪烁，需要使用 FlashWindow 函数来实现，FlashWindow 函数原型如下：

```
BOOL FlashWindow(BOOL bInvert);
```

参数说明：

- bInvert：为 TRUE 时闪烁，为 FALSE 时不闪烁。

实现过程

- 新建一个基于对话框的应用程序，将窗体标题改为“闪烁的窗体标题栏”。
- 主要程序代码如下：

```
void CSSdbIDlg::OnTimer(UINT nIDEvent)
```

```

{
    FlashWindow(true);           //设置标题栏闪烁
    CDialog::OnTimer(nIDEvent);
}

```

举一反三

根据本实例，读者可以：

- 实现窗体标题栏广告；



图-1.49 闪烁的窗体标题栏

- 实现闪烁的窗体。

实例 039 隐藏和显示标题栏

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\01\039

实例说明

窗体的标题栏与窗体标题栏上的按钮一样是可选的，通过修改窗体的样式就可以显示或隐藏这些窗体元素，结果如图 1.50 所示。

技术要点

对于窗体标题栏的显示或隐藏也就是通过改变窗体的样式实现的，实现样式的修改可以使用 API 函数 SetWindowLong 通过设置窗体的属性来实现。语法格式如下：

```
LONG SetWindowLong( HWND hWnd, int nIndex, LONG dwNewLong);
```

参数说明：

- hWnd：设置窗体属性的窗体句柄。
- nIndex：窗体属性的值。
- dwNewLong：窗体属性的值。

实现过程

(1) 新建一个基于对话框的应用程序。

(2) 为“隐藏标题栏”按钮添加单击事件，实现窗体标题栏的隐藏，代码如下：

```
void CTitleDlg::OnButton1()
{
    LONG lStyle = ::GetWindowLong(this->m_hWnd, GWL_STYLE); //获取窗体样式
    ::SetWindowLong(this->m_hWnd, GWL_STYLE, lStyle & ~WS_CAPTION); //取消窗体标题栏
    ::SetWindowPos(this->m_hWnd, NULL, 0, 0, 0, 0, SWP_NOSIZE
        | SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE | SWP_FRAMECHANGED); //重新设置窗体大小
}
```

(3) 为“显示标题栏”按钮添加单击事件，实现窗体标题栏的显示，代码如下：

```
void CTitleDlg::OnButton2()
{
    LONG lStyle = ::GetWindowLong(this->m_hWnd, GWL_STYLE); //获取窗体样式
    ::SetWindowLong(this->m_hWnd, GWL_STYLE, lStyle | WS_CAPTION); //添加窗体标题栏
    ::SetWindowPos(this->m_hWnd, NULL, 0, 0, 0, 0, SWP_NOSIZE
        | SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE | SWP_FRAMECHANGED); //重新设置窗体样式
}
```

举一反三

根据本实例，读者可以：

- 拖动没有标题栏的窗体。

实例 040 禁用标题栏上的最大化、最小化或关闭按钮

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\01\040

实例说明

在窗体的标题栏上都设置了“关闭”、“最大化”、“最小化”按钮。如果需保持窗体的状态，即不使窗体最大化、最小化或关闭，则可以将这 3 个按钮设置为“禁用”。运行程序，结果

如图 1.51 所示。

技术要点

本实例使用 API 函数 `GetWindowLong` 和 `SetWindowLong` 改变窗口风格，设置最大化和最小化按钮是否有效，使用 `SetWindowPos` 重画标题栏，使用 `GetSystemMenu` 函数获得系统菜单，使用 `GetMenuItemID` 函数获得关闭按钮的 ID，再使用 `EnableMenuItem` 函数设置关闭按钮是否有效。各函数的用法如下所示。

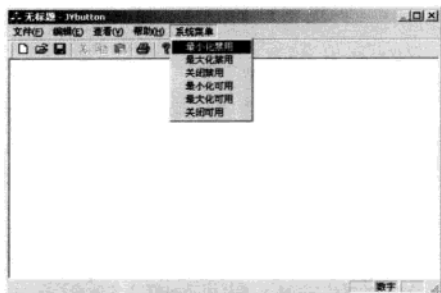


图 1.51 禁用标题栏按钮信息的菜单

(1) `GetWindowLong` 函数。该函数获得有关指定窗口的信息。函数原型如下：

```
LONG GetWindowLong(HWND hWnd, int nIndex);
```

参数说明：

- `hWnd`：窗口句柄及间接给出的窗口所属的窗口类。
- `nIndex`：指定要获得值的大于等于 0 的值的偏移量。

(2) `SetWindowLong` 函数。该函数改变指定窗口的属性。函数原型如下：

```
LONG SetWindowLong(HWND hWnd, int nIndex, LONG dwNewLong);
```

参数说明：

- `hWnd`：窗口句柄及间接给出的窗口所属的类。
- `nIndex`：指定将要设定的大于等于 0 的偏移值。
- `dwNewLong`：指定的替换值。

(3) `GetSystemMenu` 函数。该函数允许应用程序为复制或修改而访问窗口菜单（系统菜单或控制菜单）。函数原型如下：

```
CMenu* GetSystemMenu(BOOL bRevert);
```

参数说明：

- `bRevert`：指定将执行的操作。如果此参数为 `FALSE`，`GetSystemMenu` 返回当前使用窗口菜单的备份的句柄。该备份初始时与窗口菜单相同，但可以被修改。如果此参数为 `TRUE`，`GetSystemMenu` 重置窗口菜单到默认状态。如果存在先前的窗口菜单，则将被销毁。

(4) `GetMenuItemID` 函数。该函数返回位于菜单中指定位置处的项目的菜单 ID。函数原型如下：

```
UINT GetMenuItemID(int nPos);
```

参数说明：

- `nPos`：指定项目在菜单中的位置。

(5) `EnableMenuItem` 函数。该函数使指定的菜单项有效、无效或变灰。函数原型如下：

```
UINT EnableMenuItem(UINT nIDEnableItem, UINT nEnable);
```

参数说明：

- `nIDEnableItem`：指定将使其有效、无效或变灰的菜单项，按参数 `uEnable` 确定含义。此参数可指定菜单条、菜单或子菜单里的菜单项。
- `uEnable`：指定控制参数 `uIDEnableItem` 如何解释的标志，指示菜单项有效、无效或者变灰。此参数必须是 `MF_BYCOMMAND` 或 `MF_BYPOSITION`，`MF_ENABLED` 和 `MF_DISABLE` 或 `MF_GRAYED` 的组合。

实现过程

(1) 新建一个基于单文档的应用程序。

(2) 在 `ResourceView` 视图中展开 `Menu` 文件夹，双击 `IDR_MAINFRAME` 项为菜单资源添加菜单项。

(3) 在类向导中为添加的菜单添加单击事件。

(4) 主要程序代码如下：

```
void CMainFrame::OnMenudismiss()           //禁用最小化按钮
{
    Style = ::GetWindowLong(m_hWnd,GWL_STYLE); //获得窗口风格
    Style &= ~(WS_MINIMIZEBOX);
    ::SetWindowLong(m_hWnd,GWL_STYLE,Style);
    GetWindowRect(&Rect); //获得窗口区域
    //重画窗口边框
    ::SetWindowPos(m_hWnd,HWND_TOP,Rect.left,Rect.top,Rect.Width(),Rect.Height(),SWP_DRAWFRAME);
}

void CMainFrame::OnMenuismax()             //禁用最大化按钮
{
    Style = ::GetWindowLong(m_hWnd,GWL_STYLE); //获得窗口风格
    Style &= ~(WS_MAXIMIZEBOX);
    ::SetWindowLong(m_hWnd,GWL_STYLE,Style); //设置新的风格
    GetWindowRect(&Rect); //获得窗口区域
    //重画窗口边框
    ::SetWindowPos(m_hWnd,HWND_TOP,Rect.left,Rect.top,Rect.Width(),Rect.Height(),SWP_DRAWFRAME);
}

void CMainFrame::OnMenuisclose()           //禁用关闭按钮
{
    CMenu *pMenu = GetSystemMenu(false); //获得系统菜单
    UINT ID = pMenu->GetMenuItemID(pMenu->GetMenuItemCount()-1); //获得关闭按钮ID
    pMenu->EnableMenuItem(ID,MF_GRAYED); //使关闭按钮无效
}

void CMainFrame::OnMenuablemin()           //使最小化按钮有效
{
    // TODO: Add your command handler code here

    Style = ::GetWindowLong(m_hWnd,GWL_STYLE); //获得窗口风格
    Style |= WS_MINIMIZEBOX;
    ::SetWindowLong(m_hWnd,GWL_STYLE,Style); //设置新的风格
    GetWindowRect(&Rect); //获得窗口区域
    //重画窗口边框
    ::SetWindowPos(m_hWnd,HWND_TOP,Rect.left,Rect.top,Rect.Width(),Rect.Height(),SWP_DRAWFRAME);
}

void CMainFrame::OnMenuablemax()           //使最大化按钮有效
{
    Style = ::GetWindowLong(m_hWnd,GWL_STYLE); //获得窗口风格
    Style |= WS_MAXIMIZEBOX;
    ::SetWindowLong(m_hWnd,GWL_STYLE,Style); //设置新的风格
    GetWindowRect(&Rect); //获得窗口区域
    //重画窗口边框
    ::SetWindowPos(m_hWnd,HWND_TOP,Rect.left,Rect.top,Rect.Width(),Rect.Height(),SWP_DRAWFRAME);
}

void CMainFrame::OnMenuableclose()         //使关闭按钮有效
{
    CMenu *pMenu = GetSystemMenu(false); //获得系统菜单
    UINT ID = pMenu->GetMenuItemID(pMenu->GetMenuItemCount()-1); //获得关闭按钮ID
    pMenu->EnableMenuItem(ID,MF_ENABLED); //使关闭按钮可用
}
```

举一反三

根据本实例，读者可以：

- 隐藏/显示标题栏上的各个按钮；
- 绘制标题栏按钮。

1.11 窗体形状及应用

将对话框以不规则的颜色显示在桌面上，可以给人一种新鲜的感觉。本节中的实例实现了特殊的对话框效果，增加了程序的视觉效果。

实例 041 半透明窗体

本实例是一个提高效率的程序

实例位置: 光盘\mingrisoft\01\041

实例说明

很多专业软件在启动前都会显示一个说明该软件信息或用途的窗口, 这些窗口很多都是非常漂亮的半透明窗体。本实例实现了一个半透明的窗体, 效果如图 1.52 所示。

技术要点

要实现窗体的半透明效果, 首先需要窗体具有 0x80000 值的扩展风格, 然后调用 User32 动态库中的 SetLayeredWindowAttributes 函数设置半透明窗体。在 Visual C++ 中, SetLayeredWindowAttributes 函数并没有被直接封装, 需要用户手动从 User32 动态库中导入。

使窗体具有 0x80000 值的扩展风格很容易, 可以调用 SetWindowLong API 函数实现。该函数语法如下:

```
LONG SetWindowLong(HWND hWnd, int nIndex, LONG dwNewLong);
```

参数说明:

- hWnd: 表示窗口句柄。
- nIndex: 表示修改窗口的哪一个特征, 本实例需要修改窗口的扩展风格, 因此该参数应为 GWL_EXSTYLE。
- dwNewLong: 表示窗口新的特征。

导入 SetLayeredWindowAttributes 函数, 首先需要定义一个与 SetLayeredWindowAttributes 函数具有相同函数原型的函数指针, 例如:

```
typedef BOOL (WINAPI *FSetLayeredWindowAttributes)(HWND,COLORREF,BYTE,DWORD);  
FSetLayeredWindowAttributes SetLayeredWindowAttributes;
```

然后调用 LoadLibrary 函数加载 User32 动态库, 最后调用 GetProcAddress 函数将 SetLayeredWindowAttributes 指向 User32 动态库中的 SetLayeredWindowAttributes 函数。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框类中添加一个 CFont 变量 m_font。
- (3) 在对话框的 OnInitDialog 方法中设置窗口扩展风格, 并调用 User32 动态库中的

SetLayeredWindowAttributes 函数, 代码如下:

```
SetWindowLong(GetSafeHwnd(),GWL_EXSTYLE,  
GetWindowLong(GetSafeHwnd(),GWL_EXSTYLE)|0x80000); //设置窗口扩展风格  
typedef BOOL (WINAPI *FSetLayeredWindowAttributes)(HWND,COLORREF,BYTE,DWORD);  
FSetLayeredWindowAttributes SetLayeredWindowAttributes;  
HINSTANCE hInst = LoadLibrary("User32.DLL"); //加载动态链接库  
SetLayeredWindowAttributes = (FSetLayeredWindowAttributes)  
GetProcAddress(hInst,"SetLayeredWindowAttributes"); //获得函数地址  
if (SetLayeredWindowAttributes)  
SetLayeredWindowAttributes(GetSafeHwnd(),RGB(0,0,0),128,2); //设置透明度  
FreeLibrary(hInst); //释放动态链接库  
m_font.CreateFont(18,16,0,0,600,0,0,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,  
CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,FF_SCRIPT,"宋体"); //创建字体
```

举一反三

根据本实例, 读者可以:

- 设计半透明渐显窗体。



图 1.52 半透明窗体

实例 042 创建字型窗体

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\01\042

实例说明

在启动大型的应用程序时,往往要等待一段时间,这样会给用户一种程序可能没有运行起来的错觉,如果加上启动界面就可以消除这样的错觉了。启动界面通常可以设计成字形窗体等样式,本实例将创建一个字型窗体,实例运行结果如图 1.53 所示。

技术要点

明日科技

图 1.53 创建字型窗体

要设计字型窗体可以利用设备上下文 CDC 类的通道方法实现,包括 BeginPath、EndPath 和 TextOut 等方法。

(1) BeginPath 方法。BeginPath 方法用于在设备环境中打开路径,语法如下:

```
BOOL BeginPath( );
```

(2) EndPath 方法。EndPath 方法用于在设备环境中关闭路径,语法如下:

```
BOOL EndPath( );
```

(3) TextOut 方法。TextOut 方法用于输出文本,语法如下:

```
virtual BOOL TextOut( int x, int y, LPCTSTR lpszString, int nCount );
```

```
BOOL TextOut( int x, int y, const CString& str );
```

参数说明:

- x、y: 指定文本起点的横坐标和纵坐标。
- lpszString: 要绘制的字符串的指针。
- nCount: 字符串中的字节数。
- str: 包含字符的 CString 对象。

实现过程

(1) 新建一个基于对话框的应用程序。

(2) 主要程序代码如下:

```
BOOL CFontWindowDlg::OnInitDialog()  
{  
    //此处代码省略  
    CDC* pDC = GetDC();           //获得设备上下文  
    font.CreatePointFont(800,"宋体",pDC); //创建字体  
    pDC->SelectObject(&font);       //选入字体  
    pDC->BeginPath();               //打开路径  
    pDC->SetBkMode(TRANSPARENT);    //设置背景透明  
    pDC->TextOut(20,20,"明日科技"); //输出字符串  
    pDC->EndPath();                 //关闭路径  
    HRGN rgn;  
    rgn = PathToRegion(pDC->m_hDC); //获得路径区域  
    SetWindowRgn(rgn,TRUE);        //设置窗体区域  
    pDC->StrokePath();               //使用当前画笔绘制路径  
    font.DeleteObject();  
    return TRUE;  
}
```

举一反三

根据本实例,读者可以:

- 实现透明窗体;
- 实现不规则窗体。

实例 043 换肤窗体

本实例是人性化的程序

实例位置: 光盘\mingrisoft\01\043

实例说明

在程序中实现界面换肤的方式有很多种, 有的使用第 3 方或自定义控件、有的使用组件库, 而在本节中, 笔者将利用钩子技术实现界面换肤, 效果如图 1.54 和图 1.55 所示。

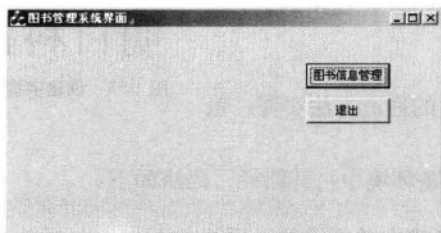


图 1.54 普通界面

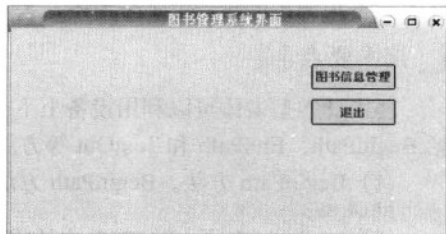


图 1.55 换肤后的界面

技术要点

在本示例中使用的主要技术有安装和卸载钩子、修改控件的窗口函数、为控件关联附加数据结构等。下面逐一进行介绍。

(1) 安装和卸载钩子

为了在程序中安装一个钩子, 需要使用 SetWindowsHookEx 函数。该函数语法如下:

```
HHOOK SetWindowsHookEx(int idHook, HOOKPROC lpfn, HINSTANCE hMod, DWORD dwThreadId);
```

参数说明:

- idHook: 表示安装的钩子类型。如果为 WH_CALLWNDPROC, 表示安装一个监视窗口过程的钩子。这样当消息被发送到目标窗口过程之前, 将被发送到钩子函数中。
- lpfn: 表示钩子函数。
- hMod: 表示包含钩子函数的实例句柄, 可以设置为 NULL。
- dwThreadId: 表示钩子函数关联的线程 ID, 如果设置为 0, 则钩子函数关联桌面中所有运行的线程。

返回值: 如果函数执行成功, 返回值是钩子函数句柄, 如果函数执行失败, 返回值为 NULL。

如果应用程序安装了一个钩子, 在应用程序退出时还需要卸载钩子。可以使用 UnhookWindowsHookEx 函数来卸载之前安装的钩子。该函数语法如下:

```
BOOL UnhookWindowsHookEx(HHOOK hhk);
```

参数说明:

- hhk: 表示欲卸载的钩子句柄, 通常为 SetWindowsHookEx 函数的返回值。

返回值: 如果函数执行成功, 返回值为 TRUE, 否则为 FALSE。

(2) 修改控件窗口函数

为了实现界面换肤, 需要修改控件默认的窗口函数, 将其替换为我们自定义的窗口函数。在程序中使用 SetWindowLong 函数来设置窗口函数, 该函数语法如下:

```
LONG SetWindowLong(HWND hWnd, int nIndex, LONG dwNewLong);
```

参数说明:

- hWnd: 表示设置窗口函数的窗口句柄。
- nIndex: 表示设置的窗口选项, 如果为 DWL_DLGPROC, 表示设置控件的窗口函数。

- dwNewLong: 表示设置的选项值, 如果 nIndex 为 DWL_DLGPROC, 则该参数表示窗口新的窗口函数。

返回值: 如果函数执行成功, 返回值是窗口之前的选项值。例如如果 nIndex 为 DWL_DLGPROC, 则函数的返回值为窗口原来的窗口函数。如果函数执行失败, 返回值为 0。

(3) 控件关联附加数据结构

在设计界面换肤时, 程序中为不同类型的控件关联了不同的数据结构信息, 例如为编辑框控件关联 CDrawEdit 结构, 为按钮控件关联 CDrawButton 结构等。为控件关联附加的结构信息, 可以使用 SetWindowLong 函数, 只要将该函数的第 2 个参数即 nIndex 设置为 GWL_USERDATA, 即表示为控件关联一个用户定义的数据结构, 此时, 函数的第 3 个参数表示附加的数据结构信息。例如:

```
pButton = new CDrawButton;
pButton->m_OldProc = WndProc;
SetWindowLong(hWnd, GWL_USERDATA, (long)pButton);
```

实现过程

- (1) 创建一个 MFC 动态链接库, 工程名为 “WndDll”。
- (2) 向工程导入一些位图文件。
- (3) 在工程中定义一个关联编辑框控件的数据结构。实现代码如下:

```
class CDrawEdit
{
public:
    WNDPROC      m_OldProc;           //记录编辑框的窗口函数
    int          m_Flag;
public:
    CDrawEdit()
    {
        m_OldProc = NULL;
        m_Flag = 0;
    }
    HBRUSH CtlColor(HWND hWnd, HDC hDC, UINT nCtlColor)
    {
        CDC* dc = CDC::FromHandle(hDC);           //获取画布对象
        CRect rect;
        ::GetClientRect(hWnd, rect);              //获取客户区域
        rect.InflateRect(1, 1, 1, 1);              //将客户区域增大一个像素
        CPen pen(PS_SOLID, 1, RGB(0, 255, 0));    //创建画笔
        dc->SelectObject(&pen);
        CBrush brush(RGB(0, 255, 0));              //创建画刷
        dc->FrameRect(rect, &brush);               //绘制边框
        return brush;
    }
};
```

- (4) 定义一个编辑框窗口函数, 在钩子函数中, 将使用该函数来替换掉编辑框控件默认的窗口函数。实现代码如下:

```
LRESULT __stdcall EditWindowProc(HWND hWnd, UINT Msg, WPARAM wParam,
                                  LPARAM lParam)
{
    CPoint pt;
    CDrawEdit *pEdit=(CDrawEdit*)GetWindowLong(hWnd, GWL_USERDATA);
    switch (Msg)
    {
        case WM_PAINT:
        {
            pEdit->CtlColor(hWnd, ::GetDC(hWnd), 0);
            break;
        }
        case WM_DESTROY:
        {
            WNDPROC procOld=pEdit->m_OldProc;
            SetWindowLong(hWnd, GWL_WNDPROC, (long)procOld); //恢复原来的窗口函数
            CWnd* pWnd = ::CWnd::FromHandle(hWnd);           //将按钮对象与句柄分离
            if (pWnd)
            {
                pWnd->Detach();
            }
        }
    }
}
```



```

    }
    pEdit->m_Flag = 1;
    return 1;
}
default:
{
    break;
}
}
return CallWindowProc(pEdit->m_OldProc, hWnd, Msg, wParam, lParam);
}

```

举一反三

根据本实例，读者可以：

- 实现透明窗体。

1.12 通用对话框的应用

通用对话框是由操作系统提供的任何应用程序都可获得的对话框，使用这些对话框，可以为用户提供他们所期望的一致性的标准界面。在 Visual C++ 中，这些对话框被封装在 CCommonDialog 及其派生类中。本书中的实例实现了通用对话框的应用。

实例 044 打开位图预览对话框

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\01\044

实例说明

在使用打开对话框选择图片时，如果能够对图片进行预览就可以让用户更加快捷地选择想要的图片。运行本实例，单击“打开”按钮，在弹出的“打开”对话框中选择一个 BMP 位图资源，用户可以在“打开”对话框的下方看到该位图的预览图像以及图像尺寸、文件大小和创建时间等信息，效果如图 1.56 所示。

技术要点

文件对话框 CFileDialog 有一个数据成员 m_ofn，该成员为 OPENFILENAME 结构变量。OPENFILENAME 结构包含了一组成员，其中 Flags 是初始化对话框的一组标记，lpTemplateName 用于提供文件对话框的子对话框窗口，如果 Flags 中包含 OFN_EXPLORER 标记，系统将创建一个标准对话框的子对话框。如果用户想要设计自己的文件对话框，可以创建一个新的对话框，在新的对话框中添加控件，然后将其赋给 lpTemplateName 成员。在这里有一点需要注意，如果文件对话框的子窗口采用了用户提供的对话框，则系统为文件对话框提供的标准控件也将显示在对话框中。因此，用户在向对话框中添加控件时，需要调整控件的位置。可以采用这样一种方式：在对话框中放置一个 CStatic 控件，将其 ID 设置为“stc32”。“stc32”是系统预定的标记，它用来确定标准文件对话框中控件显示的位置，用户可以依据它来放置自己添加的控件。

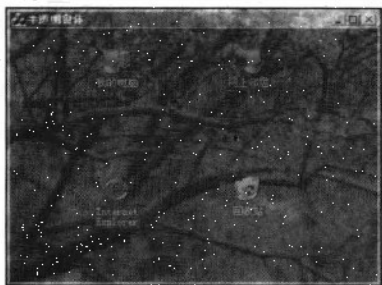


图 1.56 半透明窗体

实现过程

- (1) 新建一个基于对话框的应用程序。

- (2) 向对话框中添加一个按钮控件, 设置其 Caption 属性为“打开”。
- (3) 新建一个对话框资源, 向新建的对话框中添加 1 个图片控件和 3 个静态文本控件。
- (4) 以 CFileDialog 类为基类派生一个 CNewFileDialog 类。
- (5) 在 CNewFileDialog 类的头文件中声明变量, 代码如下:

```
CStatic m_Bitmap;           //图片控件对应变量
CStatic m_Measure;          //图像尺寸控件对应变量
CStatic m_Size;             //文件大小控件对应变量
CStatic m_Time;             //创建时间控件对应变量
```

- (6) 在 CNewFileDialog 类的构造函数中设置 m_ofn 成员, 代码如下:

```
CNewFileDialog::CNewFileDialog(BOOL bOpenFileDialog, LPCTSTR lpszDefExt, LPCTSTR lpszFileName,
    DWORD dwFlags, LPCTSTR lpszFilter, CWnd* pParentWnd):
    CFileDialog(bOpenFileDialog, lpszDefExt, lpszFileName, dwFlags, lpszFilter, pParentWnd)
{
    m_ofn.Flags = (OFN_EXPLORER| OFN_ENABLETEMPLATE| OFN_ENABLEHOOK);
    //将对话框资源赋给lpTemplateName成员
    m_ofn.lpszTemplateName = MAKEINTRESOURCE(IDD_NEWFILE_DIALOG);
}
```

- (7) 处理 CNewFileDialog 类的 WM_INITDIALOG 消息, 将声明的成员关联到对话框中的静态文本控件, 代码如下:

```
BOOL CNewFileDialog::OnInitDialog()
{
    CFileDialog::OnInitDialog();
    m_Bitmap.SubclassDlgItem(IDC_STATIC1,this);    //关联图片控件
    m_Measure.SubclassDlgItem(IDC_STATIC2,this);   //关联图像尺寸控件
    m_Size.SubclassDlgItem(IDC_STATIC3,this);      //关联文件大小控件
    m_Time.SubclassDlgItem(IDC_STATIC4,this);      //关联创建时间控件
    return TRUE;
}
```

- (8) 在 CNewFileDialog 类中, 添加 OnFileNameChange 虚拟方法, 在文件改变时, 判断选择的文件是否是位图, 如果是, 则将其显示在图片控件中, 并将其信息显示在静态文本控件中, 代码如下:

```
void CNewFileDialog::OnFileNameChange()
{
    CString exp;
    exp=GetFileExt();
    exp.MakeUpper();           //在比较扩展名时不区分大小写
    if(exp == "BMP")           //显示位图
    {
        CFile file;            //声明CFile类对象
        if(!file.Open(GetPathName(),CFile::modeRead) ) //打开文件
            return;
        BITMAPFILEHEADER bmfHeader;
        if(file.Read((LPSTR)&bmfHeader,sizeof(bmfHeader)) != sizeof(bmfHeader)) //读位图文件头信息
            return;
        BITMAPINFOHEADER bmiHeader;
        if(file.Read((LPSTR)&bmiHeader, sizeof(bmiHeader)) != sizeof(bmiHeader)) //读位图头信息
            return;
        //获得大小信息, 并显示
        int bmWidth = bmiHeader.biWidth;           //获得位图宽度
        int bmHeight = bmiHeader.biHeight;          //获得位图高度
        file.Close();                               //关闭文件
        int x=150,y=150;
        if(bmWidth < 150)                           //如果文件宽度小于150像素
            x = bmWidth;                           //设置其显示宽度
        if(bmHeight < 150)                          //如果文件高度小于150像素
            y = bmHeight;                          //设置其显示高度
        m_Bitmap.SetBitmap((HBITMAP)LoadImage(NULL,GetPathName(),
            IMAGE_BITMAP,x,y,LR_LOADFROMFILE));    //显示位图
        CFileStatus status;
        CFile::GetStatus(GetPathName(),status);    //获得位图信息
        CString measure,size,time;
        measure.Format("图像尺寸: %d*%d",bmWidth,bmHeight); //格式化图像尺寸
    }
```

```

size.Format("文件大小: %dK", status.m_size/1024);           //格式化文件大小
time.Format("创建时间: %s", status.m_ctime.Format("%Y年%m月%d日 %H:%M:%S")); //格式化创建时间
m_Measure.SetWindowText(measure);                          //显示图像尺寸
m_Size.SetWindowText(size);                                //显示文件大小
m_Time.SetWindowText(time);                                //显示创建时间
    }
}

```

(9) 处理主窗口中“打开”按钮的单击事件, 在该事件的处理函数中调用“打开”对话框, 代码如下:

```

void CBmpPreviewDlg::OnOpen()
{
    CNewFileDialog dlg(true, NULL, NULL, OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT |
        OFN_EXPLORER | OFN_ENABLETEMPLATE, "All Files (*.BMP)*.BMP|"); //构造打开对话框
    dlg.DoModal(); //显示打开对话框
}

```

举一反三

根据本实例, 读者可以:

- 隐藏打开对话框中默认的控件。

实例 045 打开 Windows 新型对话框

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\01\045

实例说明

在 Windows 2000 及以后版本的操作系统中提供了新型的打开对话框, 在程序中如何调用新型的打开对话框呢? 本实例将实现调用 Windows 新型打开对话框的功能, 实例运行结果如图 1.57 所示。

技术要点

可以使用 `GetOpenFileName` 函数实现 Windows 新型打开对话框的调用, 在使用 `GetOpenFileName` 函数时, 需要设置 `OPENFILENAME` 结构的参数。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加一个按钮控件, 设置控件 ID 和 Caption 属性。
- (3) 主要程序代码如下:

```

void CWindowsDialogDlg::OnButtonopen()
{
    OPENFILENAME fopt; //声明OPENFILENAME结果
    memset(&fopt, 0, sizeof(fopt)); //设置结构大小
    fopt.lStructSize = sizeof(OPENFILENAME); //设置结构大小
    fopt.Flags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT; //设置标记
    fopt.hwndOwner = GetSafeHwnd(); //窗口句柄
    fopt.lpstrFilter = "All Files (*.*)"; //过滤字符串
    GetOpenFileName(&fopt); //调用打开对话框
}

```

举一反三

根据本实例, 读者可以:



图 1.57 Windows 新型打开对话框

● 实现新型保存对话框。

实例 046 同时选择多个文件

本实例是人性化的程序

实例位置：光盘\mingrisoft\01\046

实例说明

在使用“打开”对话框时，可以一次性选择多个文件，这样可以方便用户的操作，本实例窗体实现了同时选择多个文件的功能。运行程序，单击“打开”按钮，可以在弹出的“打开”对话框中同时选择多个文件，如图 1.58 所示。

技术要点

使用“打开”对话框同时选择多个文件时，需要在构造“打开”对话框时设置 `OFN_ALLOWMULTISELECT` 属性，构造“打开”对话框通过 `CFileDialog` 构造函数来实现，语法如下：

```
CFileDialog( BOOL bOpenFileDialog, LPCTSTR lpszDefExt = NULL, LPCTSTR lpszFileName = NULL, DWORD dwFlags =  
OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, LPCTSTR lpszFilter = NULL, CWnd* pParentWnd = NULL );
```

参数说明：

- `bOpenFileDialog`：如果值为 `TRUE`，构造“打开”对话框；为 `FALSE`，构造“另存为”对话框。
- `lpszDefExt`：用于确定文件默认的扩展名，如果为 `NULL`，没有扩展名被插入到文件名中。
- `lpszFileName`：确定编辑框中初始化时的文件名称，如果为 `NULL`，编辑框中没有文件名称。
- `dwFlags`：用于自定义文件对话框。
- `lpszFilter`：用于指定对话框过滤的文件类型。
- `pParentWnd`：标识文件对话框的父窗口指针。

注意：`lpszFilter` 参数格式，文件类型说明和扩展名间用“|”分隔，每种文件类型间用“|”分隔，末尾用“||”结束。

在获得“打开”对话框选择的文件路径时，需要使用 `GetStartPosition` 函数和 `GetNextPathName` 函数。

`GetStartPosition` 函数用于获取列表中第一个文件路径名的位置，语法如下：

```
POSITION GetStartPosition( ) const;
```

`GetNextPathName` 函数用于从对话框所选的组中获取下一个文件名，文件名路径包括文件标题加上整个目录路径，语法如下：

```
CString GetNextPathName( POSITION& pos ) const;
```

由前面 `GetNextPathName` 或 `GetStartPosition` 函数调用返回的一个 `POSITION` 值。如果达到列表尾，则为 `NULL`。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在程序中添加一个列表框控件和一个按钮控件。
- (3) 主要程序代码如下：

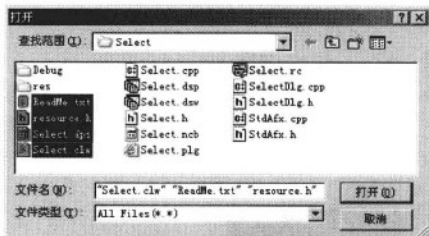


图 1.58 同时选择多个文件


```
void CSelectDlg::OnButselect()
{
    CFileDialog dlg(TRUE,NULL,NULL,OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT
        |OFN_ALLOWMULTISELECT,"All Files(*.*)",AfxGetMainWnd()); //构造文件打开对话框
    CString strPath=""; //声明变量
    if(dlg.DoModal() == IDOK) //判断是否按下"打开"按钮
    {
        POSITION m_Position = dlg.GetStartPosition(); //获取第一个文件名位置
        while(m_Position != NULL)
        {
            strPath = dlg.GetNextPathName(m_Position); //获得下一个文件路径
            m_List.InsertString(m_List.GetCount(),strPath); //向列表框中插入文件路径
        }
    }
}
```

举一反三

根据本实例，读者可以：

- 设置文件保存对话框。

实例 047 文本替换对话框

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\01\047

实例说明

使用“查找/替换”对话框可以进行查找和替换操作，CFindReplaceDialog 类对“查找/替换”对话框进行了封装，下面通过实例来制作一个文本替换对话框，实例运行结果如图 1.59 所示。

技术要点

要使用文本替换对话框，首先要了解 CFindReplaceDialog 类中封装的各个函数的功能，CFindReplaceDialog 类中的常用函数如表 1.1 所示。

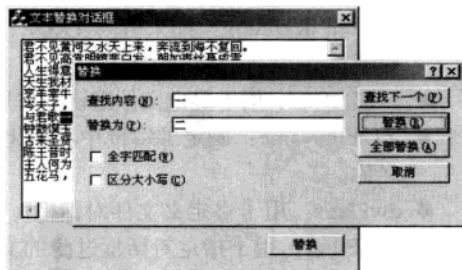


图 1.59 文本替换对话框

表 1.1

CFindReplaceDialog 类常用函数表

函 数 名	描 述
Create	创建文本替换对话框
FindNext	用于确定是否需要查找下一个字符串
GetNotifier	用于获取查找、替换对话框指针
GetFindString	用于获取默认的查找字符串
GetReplaceString	用于获取默认的替换字符串
ReplaceAll	用于确定是否用户想要替换所有的字符串
ReplaceCurrent	用于确定是否用户想要替换当前选中的字符串
SearchDown	用于确定用户是否想要向下查找字符串

实现过程

- (1) 新建一个基于对话框的应用程序。

(2) 向对话框中添加一个编辑框控件和一个按钮控件, 设置编辑框属性, 使其可以多行显示文本, 并且具有滚动条。

(3) 在对话框头文件中声明变量, 代码如下:

```
CFindReplaceDialog* dlg;           //声明“查找/替换”对话框指针
int nIndex;                        //存储查找字符串的起始位置
BOOL degree;                       //判断是否为第一次替换的变量
BOOL find;                          //判断是否进行查找的变量
```

(4) 定义一个新消息 WM_FINDMESSAGE, 代码如下:

```
static UINT WM_FINDMESSAGE = RegisterWindowMessage(FINDMSGSTRING);
```

(5) 在对话框的头文件和消息映射部分分别添加函数定义和映射宏, 代码如下:

```
afx_msg long OnFindReplace(WPARAM wParam, LPARAM lParam);
ON_REGISTERED_MESSAGE(WM_FINDMESSAGE, OnFindReplace)
```

(6) 处理 WM_FINDMESSAGE 消息, 在该消息的响应函数中实现查找和替换操作, 代码如下:

```
long CReplaceDialogDlg::OnFindReplace(WPARAM wParam, LPARAM lParam)
{
    CString strText, repText;           //声明字符串变量
    strText = dlg->GetFindString();      //获得查找字符串
    CString str;                         //声明字符串变量
    m_Edit.GetWindowText(str);          //获得编辑框中的文本
    if(dlg->ReplaceCurrent())             //判断是否进行替换
        find = FALSE;                   //进行替换
    else
        find = TRUE;                    //进行查找
    int len;                             //声明整型变量
    if(find)                             //判断是查找还是替换
    {
        len = strText.GetLength();       //获得要查找字符串的长度
    }
    else
    {
        if(dlg->ReplaceAll())             //判断是否全部替换
        {
            repText = dlg->GetReplaceString(); //获得替换字符串
            len = repText.GetLength(); //获得替换字符串长度
            str.Replace(strText, repText); //使用替换字符串替换查找字符串
            m_Edit.SetWindowText(str); //将替换后的字符串显示在编辑框中
        }
        else
        {
            CString left, right;          //声明字符串变量
            int num = strText.GetLength(); //获得查找字符串的长度
            int strnum = str.GetLength(); //获得编辑框中文本长度
            int index;                     //声明整型变量
            if(!degree)                   //判断是否为第一次替换
                index = str.Find(strText, nIndex); //获得查找字符串在编辑框文本中的位置
            else if(nIndex-2 >= 0)         //判断起始查找位置是否小于0
                index = str.Find(strText, nIndex-2); //获得查找字符串在编辑框文本中的位置
            else
            {
                nIndex = 2;                //设置起始查找位置
                return 1;
            }
            degree = TRUE;
            left = str.Left(index);        //获得替换字符串左侧的字符串
            right = str.Right(strnum-index-num); //获得替换字符串右侧的字符串
            repText = dlg->GetReplaceString(); //获得替换字符串
            len = repText.GetLength(); //获得替换字符串长度
            str = left + repText + right; //组合成新的字符串
            m_Edit.SetWindowText(str); //在编辑框中显示新的字符串
        }
    }
    int index = str.Find(strText, nIndex); //获得查找字符串在编辑框文本中的位置
    m_Edit.SetSel(index, index+len); //选中查找或替换的字符串
}
```

```
nindex = index + len;           //设置起始查找位置  
m_Edit.SetFocus();             //编辑框获得焦点  
return 0;  
}
```

(7) 为“替换”按钮处理单击事件，创建“替换”对话框的代码如下：

```
void CReplaceDialogDlg::OnReplace() //“替换”按钮单击事件响应函数  
{  
    dlg = new CFindReplaceDialog;    //为“查找/替换”对话框指针赋值  
    dlg->Create(FALSE, NULL);        //创建“替换”对话框  
    dlg->ShowWindow(SW_SHOW);       //显示“替换”对话框  
}
```

举一反三

根据本实例，读者可以：

- 设计文件查找对话框。

实例 048 字体选择对话框

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\01\048

实例说明

CFontDialog 类封装了 Windows 字体对话框。用户可以从系统安装的字体列表中选择要用的字体，同时，在“字体”对话框中还可以设置字体大小、颜色、效果、字符集等属性。下面通过实例来演示字体选择对话框，实例运行结果如图 1.60 所示。

技术要点

要使用字体对话框就要了解 CFontDialog 类封装的功能，字体对话框的常用函数如表 1.2 所示。

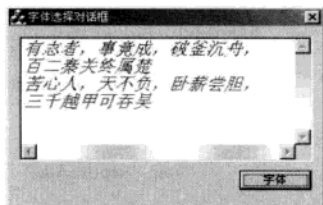


图 1.60 字体选择对话框

表 1.2 字体对话框常用函数表

函 数 名	描 述
CFontDialog	构造函数
DoModal	用于显示字体对话框，供用户设置字体
GetCurrentFont	用于获取当前的字体
GetFaceName	用于获取字体对话框中选择的字体名称
GetStyleName	用于返回字体对话框中选择的字体风格名称
GetSize	用于获取字体的大小
GetColor	用于获取选择的字体颜色
GetWeight	用于获取字体的磅数

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加一个编辑框控件和一个按钮控件，设置编辑框控件属性，使其可以多行显示，并具有滚动条。

(3) 在对话框头文件中声明一个 CFont 对象 m_Font。

(4) 主要程序代码如下：

```
void CFontDialogDlg::OnFont()
{
    CFont* TempFont = m_Text.GetFont();           //获取编辑框当前字体
    LOGFONT LogFont;
    TempFont->GetLogFont(&LogFont);
    CFontDialog dlg(&LogFont);                     //初始化字体信息
    if(dlg.DoModal()==IDOK)
    {
        m_Font.Detach();
        LOGFONT temp;
        dlg.GetCurrentFont(&temp);                 //获取当前字体信息
        m_Font.CreateFontIndirect(&temp);          //直接创建字体
        m_Color = dlg.GetColor();                   //获得字体颜色
        m_Text.SetFont(&m_Font);                   //设置字体
    }
}
```

举一反三

根据本实例，读者可以：

- 使用字体对话框为对话框中各个控件设置不同的字体。

第 2 章 控件应用



- 按钮控件典型实例
- RichEdit 控件典型实例
- 编辑框控件典型实例
- 滚动条控件典型实例
- 静态文本控件典型实例
- 进度条控件典型实例
- 列表框控件典型实例
- 工具提示控件典型实例
- 组合框控件典型实例
- 滑块控件典型实例
- 列表视图控件典型实例
- 标签控件典型实例
- 树视图控件典型实例
- 控件数组典型实例

Visual C++

2.1 按钮控件典型实例

个性化的按钮可以美化界面,从而吸引更多的用户,本节将介绍创建各种不同风格的按钮的方法。

实例 049 AVI 动画按钮

本实例是一个人性化的实例

实例位置: 光盘\mingrisoft\02\049

实例说明

在开发程序的时候,经常会用到个性化按钮来美化程序界面,其中,能播放 AVI 动画的按钮会吸引更多年轻人的目光。运行程序,当鼠标光标在控件上方移动时,按钮将产生动画效果,效果如图 2.1 所示。

技术要点

本实例主要是通过使用动画控件和设置鼠标的消息响应来实现的。

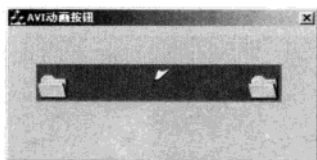


图 2.1 AVI 动画按钮

首先通过 CAnimateCtrl 类来创建和使用动画控件, CAnimateCtrl 类的方法如下所示。

(1) Open 方法。从一个文件或资源打开一个 AVI 文件,并显示第一帧。语法如下:

```
BOOL Open( LPCTSTR lpszFileName );  
BOOL Open( UINT nID );
```

参数说明:

- lpszFileName: AVI 文件名。
- nID: 资源 ID。

(2) Play 方法。播放 AVI 文件。语法如下:

```
BOOL Play( UINT nFrom, UINT nTo, UINT nRep );
```

参数说明:

- nFrom: 起始帧索引。
- nTo: 结束帧索引。
- nRep: 是否循环播放。

(3) Seek 方法。显示 AVI 文件中的指定帧,语法如下:

```
BOOL Seek( UINT nTo );
```

参数说明:

- nTo: 指定帧索引。

(4) Stop 方法。停止播放 AVI 文件,语法如下:

```
BOOL Stop();
```

(5) Close 方法。关闭已打开的 AVI 文件,语法如下:

```
BOOL Close();
```

设置鼠标的 WM_MOUSEMOVE 消息,消息响应函数原型如下:

```
afx_msg void OnMouseMove(UINT nFlags, CPoint point);
```

参数说明:

- nFlags: 指示是否按下了各种虚键。
- point: 指出光标的横坐标和纵坐标。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加一个按钮控件,用鼠标右键单击按钮控件,在弹出的菜单中选择 Properties 选项,选择 Owner Draw 属性。
- (3) 单击 Insert/Resource 菜单项,在打开的 Insert Resource 对话框中单击 Import 按钮,添加一个 AVI 文件,在弹出的 Custom Resource Type 对话框中定制新的资源类型。
- (4) 以 CButton 类为基类派生一个 CButtonAvi 类。
- (5) 重载 CButtonAvi 类的 DrawItem 虚方法,在该方法中创建播放 AVI 的动画控件,代码如下:

```
void CButtonAvi::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
    CRect rect;
    GetClientRect(rect); //获得按钮的客户区域
    if (!::IsWindow(m_Animate))
    {
        m_Animate.Create(WS_CHILD | WS_VISIBLE, rect, this, 0); //创建动画控件
        m_Animate.Open(m_id); //打开AVI文件资源
        m_Animate.GetClientRect(rect); //获得动画控件的客户区域
        VERIFY(SetWindowPos(NULL, 0, 0, rect.Width()+2, rect.Height()+2, SWP_NOMOVE)); //设置按钮控件显示位置
        m_Animate.MoveWindow(rect); //移动动画控件位置
    }
    CDC* pDC = CDC::FromHandle(lpDrawItemStruct->hDC); //获得按钮控件设备上下文
    UINT State = lpDrawItemStruct->itemState; //获得按钮控件状态
    DrawButton(pDC, State, rect); //调用自定义函数绘制按钮
}
```

- (6) 添加自定义 DrawButton, 在该函数中绘制按钮控件, 代码如下:

```
void CButtonAvi::DrawButton(CDC *pDC, UINT nState, CRect rect)
{
    COLORREF UpCol, DownCol;
    if ((nState & ODS_SELECTED) == ODS_SELECTED) //选中状态
    {
        UpCol=RGB(0,0,0); //设置上边颜色为黑色
        DownCol=RGB(0,0,0); //设置下边颜色为黑色
        m_play = false;
    }
    else if ((nState & ODS_DISABLED) != ODS_DISABLED) //如果不可用
    {
        UpCol=RGB(255,255,255); //设置上边颜色为白色
        DownCol=RGB(128,128,128); //设置下边颜色为灰色
    }
    //画按钮的左边和上边
    CPen pen1, pen2;
    pen1.CreatePen(PS_SOLID, 3, UpCol); //创建画笔
    pDC->SelectObject(&pen1); //选入画笔
    pDC->MoveTo(0, rect.Height()-1);
    pDC->LineTo(0, 0);
    pDC->LineTo(rect.Width()-1, 0);
    //画按钮的右边和下边
    pen2.CreatePen(PS_SOLID, 2, DownCol); //创建画笔
    pDC->SelectObject(&pen2); //选入画笔
    pDC->MoveTo(rect.Width()-1, 0);
    pDC->LineTo(rect.Width()-1, rect.Height()-1);
    pDC->LineTo(0, rect.Height()-1);

    pen1.DeleteObject();
    pen2.DeleteObject();
}
```

- (7) 设置鼠标的 WM_MOUSEMOVE 消息, 在该消息的响应函数中捕获鼠标光标位置, 判断是否播放 AVI 文件, 代码如下:

```
void CButtonAvi::OnMouseMove(UINT nFlags, CPoint point)
{
    ClientToScreen(&point); //将鼠标光标位置转换为屏幕坐标
    CRect rc;
    GetWindowRect(rc); //获得按钮窗口的区域
    if (rc.PtInRect(point)) //判断鼠标光标是否在按钮区域内
    {
        if (::IsWindow(m_Animate) && !m_play)
```



```

{
    m_Animate.Play(0,-1,1);           //播放AVI动画
    m_play = true;
    SetCapture();                     //捕获鼠标
}
else
{
    m_play = false;
    ReleaseCapture();                 //释放鼠标
}
CButton::OnMouseMove(nFlags, point);
}

```

举一反三

根据本实例，读者可以：

- 开发 AVI 文件播放器。

实例 050 GIF 动画按钮

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\02\050

实例说明

通常情况下，动画按钮会比其他形式的按钮更加吸引人，可是 Visual C++ 中的动画控件只能显示简单的 AVI 动画，显然这并不能满足用户的要求。本实例将创建一个可以显示 GIF 动画的按钮，运行程序，效果如图 2.2 所示。



图 2.2 GIF 动画按钮

技术要点

本实例使用 API 函数 FindResource、SizeofResource 和 LoadResource 来装载 GIF 资源。

(1) FindResource 函数：该函数确定指定模块中指定类型和名称的资源所在的位置。函数原型如下：

```
HRSRC FindResource (HMODULE hModule, LPCTSTR lpName, LPCTSTR lpType);
```

参数说明：

- hModule：处理包含资源的可执行文件的模块。NULL 值指定模块句柄指向操作系统，通常情况下创建最近过程的相关位图文件。
- lpName：指定资源名称。
- lpType：指定资源类型。

(2) SizeofResource 函数：该函数返回指定资源字节数大小。函数原型如下：

```
DWORD SizeofResource (HMODULE hModule, HRSRC hResInfo);
```

参数说明：

- hModule：包含资源的可执行文件模块的句柄。
- hResInfo：资源句柄，此句柄必须由函数 FindResource 或 FindResourceEx 来创建。

(3) LoadResource 函数：该函数装载指定资源到全局存储器。函数原型如下：

```
HGLOBAL LoadResource (HMODULE hModule, HRSRC hResInfo);
```

参数说明：

- hModule：处理包含资源的可执行文件的模块句柄。若 hModule 为 NULL，系统从当前过程的模块中装载资源。
- hResInfo：将被装载资源的句柄，它必须由函数 FindResource 或 FindResourceEx 创建。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加一个按钮控件, 选择 Owner-Draw 属性。
- (3) 以 CButton 类为基类创建一个按钮类 CButtonGif。
- (4) 主要程序代码如下。

```
//显示GIF动画
static UINT GifThread(LPVOID GifDC)
{
    CDC* pDC = (CDC*)GifDC;
    HINSTANCE hAmdle = ::AfxGetResourceHandle();
    HRSRC hRsrc = ::FindResource(hAmdle, MAKEINTRESOURCE(IDR_GIF1), "GIF"); //查找资源位置
    DWORD word = ::SizeofResource(hAmdle, hRsrc); //获得资源大小
    BYTE* lpBy = (BYTE*)LoadResource(hAmdle, hRsrc); //加载资源
    BYTE* pByte[200];
    DWORD nu[200];
    int num = 0;
    DWORD firstLocation = 0;
    for(DWORD i=0; i<word; i++)
    {
        if(lpBy[i]==0x2c) //如果是图像分隔符
        {
            if(num==0)
            {
                firstLocation = i;
            }
            pGif nImage = (pGif)&lpBy[i+1];
            DWORD number = 1+sizeof(Gif);
            while(lpBy[i+number]!=0)
            {
                number = number+(DWORD)lpBy[i+number]+1;
            }
            number++;
            pByte[num] = new BYTE[number];
            for(DWORD n=0; n<number; n++)
            {
                *(BYTE*)(pByte[num]+n) = lpBy[i+n];
            }
            nu[num] = number;
            i = i+number-1;
            num++;
        }
    }
    while(1)
    {
        for(int m=0; m<num; m++)
        {
            DWORD dWord;
            VirtualProtect(lpBy, word, PAGE_READWRITE, &dWord);
            for(DWORD n=0; n<nu[m]; n++)
            {
                lpBy[firstLocation+n] = *(BYTE*)(pByte[m]+n);
            }
            VirtualProtect(lpBy, word, dWord, NULL);
            CMemFile mfile(lpBy, word); //构造驻留文件类对象
            CArchive aRc(&mfile, CArchive::load | CArchive::bNoFlushOnDelete); //构造CArchive对象
            CArchiveStream aRcstream(&aRc);
            CComQIPtr<IPicture> m_picture;
            OleLoadPicture((LPSTREAM)&aRcstream, 0, false, IID_IPicture, (void**)&m_picture);
            long x, y;
            m_picture->get_Width(&x);
            m_picture->get_Height(&y);
            CSize size(x, y);
            pDC->HIMETRICtoDP(&size);
            CRect rect;
            m_picture->Render(*pDC, 0, 0, size.cx, size.cy, 0, y, x, -y, &rect);
            Sleep(30);
        }
    }
    return 1;
}
//播放GIF动画
void CButtonGif::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
```

```
{
    CDC* pDC = this->GetDC();           //获得设备上下文指针
    if(!m_play)
    {
        AfxBeginThread(GifThread,(LPVOID)pDC); //开启线程
        m_play = true;
    }
    CRect rect(-1,-1,180,59);
    UINT State = lpDrawItemStruct->itemState; //设置按钮状态
    DrawButton(pDC,State,rect);             //绘制按钮
}
```

举一反三

根据本实例，读者可以：

- 开发 GIF 播放器。

实例 051 图文按钮

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrsoft\01\051

实例说明

通常情况下，MFC 提供的按钮 CButton 并不能显示图标。如果在应用程序的按钮中显示一个图标，将使程序更加美观。运行程序，图文按钮效果如图 2.3 所示。

技术要点

在 Visual C++ 中，可以通过改写 CButton 的 DrawItem 方法实现自定义按钮的绘制。在按钮中绘制图标主要使用了 DrawItem 方法中 lpDrawItemStruct 参数的 hDC 成员来实现。DrawItem 方法是一个虚拟方法，用于绘制控件的外观。当按钮控件包含 BS_OWNERDRAW 风格时，应用程序将自动调用 DrawItem 方法绘制按钮。语法如下：

```
virtual void DrawItem( LPDRAWITEMSTRUCT lpDrawItemStruct );
```

参数说明：

- lpDrawItemStruct：是一个 DRAWITEMSTRUCT 结构指针，其结构成员如下。
- CtlType：表示控件的类型。可选值如下。
 - ODT_BUTTON：按钮。
 - ODT_COMBOBOX：组合框。
 - ODT_LISTBOX：列表框。
 - ODT_MENU：菜单。
 - ODT_LISTVIEW：列表视图。
 - ODT_STATIC：静态控件。
 - ODT_TAB：标签控件。
- CtlID：表示控件 ID。
- ItemID：表示菜单项 ID 或列表框、组合框中的项目索引。
- ItemAction：表示绘画的动作。可选值如下。
 - ODA_DRAWENTIRE：整个控件需要被绘制时设置该标识。
 - ODA_FOCUS：控件获得或失去焦点时设置该标识。
 - ODA_SELECT：表示控件处于选中状态时设置该标识。
- ItemState：表示需要绘画的状态。可选值如下。

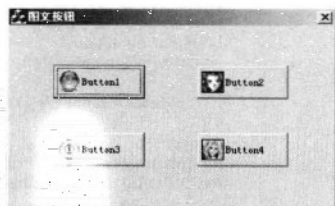


图 2.3 图文按钮

- ODS_CHECKED: 菜单项被选中。
- ODS_DISABLED: 控件不可用。
- ODS_FOCUS: 控件获得焦点。
- ODS_GRAYED: 控件处于灰色状态, 只用于菜单控件。
- ODS_SELECTED: 控件被选中。
- ODS_COMBOBOXEDIT: 组合框中编辑控件的文本被选中。
- ODS_DEFAULT: 默认状态。
- HwndItem: 表示控件的句柄。
- HDC: 控件的画布句柄。
- RclItem: 控件的矩形区域。
- ItemData: 控件的附加信息。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加 4 个按钮控件, 全都选择 Owner Draw 属性, 向工程中添加 4 个 ICO 图标资源。
- (3) 以 CButton 类为基类创建一个按钮类 CImageButton。
- (4) 主要程序代码如下:

```
void ImageButton::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
    CDC dc;
    dc.Attach(lpDrawItemStruct->hDC); //获得设备上下文
    if (m_plmagemlist)
    {
        UINT state = lpDrawItemStruct->itemState; //获取状态
        UINT action = lpDrawItemStruct->itemAction;
        //获取图像列中图像的大小
        IMAGEINFO imageinfo;
        m_plmagemlist->GetImageInfo(m_ImageIndex, &imageinfo);
        CSize imagesize;
        imagesize.cx = imageinfo.rcImage.right - imageinfo.rcImage.left;
        imagesize.cy = imageinfo.rcImage.bottom - imageinfo.rcImage.top;
        //在按钮垂直方向居中显示图标
        CRect rect;
        GetClientRect(rect);
        CPoint point;
        point.x = 5;
        point.y = (rect.Height() - imagesize.cy)/2;
        m_plmagemlist->Draw(&dc, m_ImageIndex, point, ILD_NORMAL | ILD_TRANSPARENT); //绘制图标
        //按钮被选中或者获得焦点时
        if ((state & ODS_SELECTED) || (state & ODS_FOCUS))
        {
            CRect focusRect(rect); //焦点矩形
            focusRect.DeflateRect(4, 4, 4, 4); //设置区域
            CPen pen(PS_DASHDOTDOT, 1, RGB(0, 0, 0)); //创建画笔
            CBrush brush; //创建画笔
            brush.CreateStockObject(NULL_BRUSH); //创建画刷
            dc.SelectObject(&brush); //选入画刷
            dc.SelectObject(&pen); //选入画笔
            //绘制焦点矩形
            dc.DrawFocusRect(focusRect);
            //绘制立体效果
            dc.DrawEdge(rect, BDR_RAISEDINNER | BDR_RAISEDOUTER, BF_BOTTOMLEFT | BF_TOPRIGHT);
            //获得焦点时绘制黑色边框
            dc.Draw3dRect(rect, RGB(51, 51, 51), RGB(0, 0, 0));
        }
    }
    else //默认情况下
    {
        CRect focusRect(rect); //焦点矩形
        focusRect.DeflateRect(4, 4, 4, 4); //设置区域
        CPen pen(PS_DOT, 1, RGB(192, 192, 192)); //创建画笔
        CBrush brush; //创建画笔
        brush.CreateStockObject(NULL_BRUSH); //创建画刷
    }
}
```



```

dc.SelectObject(&brush);
dc.SelectObject(&pen);
dc.Rectangle(focusRect); //绘制矩形
//绘制立体效果
dc.DrawEdge(rect,BDR_RAISEDINNER|BDR_RAISEDOUTER,BF_BOTTOMLEFT|BF_TOPRIGHT);
}
if (IsPressed) //在按钮被按下时绘制按下效果
{
    CRect focusRect1(rect);
    focusRect1.DeflateRect(4,4,4,4);
    dc.DrawFocusRect(focusRect1); //绘制焦点矩形
    dc.DrawEdge(rect,BDR_SUNKENINNER|BDR_SUNKENOUTER,BF_BOTTOMLEFT|BF_TOPRIGHT);
    dc.Draw3dRect(rect,RGB(51,51,51),RGB(0,0,0)); //绘制3D边框
}

CString text;
GetWindowText(text); //获得按钮文本
rect.DeflateRect(point.x+imagesize.cx+2,0,0,0); //设置文本显示区域
dc.SetBkMode(TRANSPARENT); //设置背景透明
dc.DrawText(text,rect,DT_LEFT|DT_SINGLELINE|DT_VCENTER); //绘制按钮文本
}
}

```

举一反三

根据本实例，读者可以：

- 实现位图按钮。

实例 052 按钮七巧板

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\02\052

实例说明

用户在使用 Visual C++ 进行程序开发时，按钮控件都是矩形按钮，可是在有些特殊的界面中，使用这种矩形按钮可能会使程序界面看起来很死板，为了能更好的和程序界面进行搭配，可以尝试创建各种不同形状的按钮。本实例就实现设计各种形状的按钮控件，实例运行效果如图 2.4 所示。

技术要点

在 Visual C++ 中，可以通过改写 CButton 的 DrawItem 方法实现自定义按钮的绘制。在绘制按钮的过程中主要使用了 Polygon 方法。该方法用于绘制多边形，语法如下：

```
BOOL Polygon( LPPOINT lpPoints, int nCount );
```

参数说明：

- lpPoints：存储多边形顶点数组的指针。
- nCount：数组中的顶点数。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 以 CButton 类为基类，派生一个 CCustomButton 类，该类用于设置按钮形状。
- (3) 向窗体中添加 6 个按钮控件，设置按钮控件的 Owner Draw 属性，为按钮控件关联

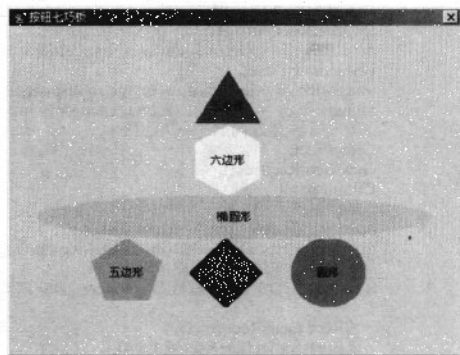


图 2.4 按钮七巧板

CCustomButton 类对象。

(4) 重写 DrawItem 方法, 在该方法中绘制按钮外观, 代码如下:

```
void CCustomButton::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
    CRect rect;
    GetClientRect(rect); //获得按钮客户区域
    CDC dc;
    dc.Attach(lpDrawItemStruct->hDC); //获得设备上下文
    int x,y,r;
    x = rect.Width()/2;
    y = rect.top;
    r = rect.Height()/2;
    double lpi=0;
    arrays[0] = CPoint(x,y); //设置多边形第一个顶点坐标
    if(m_result)
    {
        for(int i=1;i<m_num;i++) //根据多边形顶点数循环
        {
            lpi+=(2*PI/m_num); //获得每个顶点的相对角度
            if(lpi<=2*PI/4) //小于等于90度时
            {
                arrays[i] = CPoint(x+r*sin(2*i*PI/m_num),r-r*cos(2*i*PI/m_num));
            }
            if(lpi>2*PI/4 && lpi<=2*PI/2) //大于90度小于等于180度
            {
                arrays[i] = CPoint(x+r*sin(PI-2*i*PI/m_num),r+r*cos(PI-2*i*PI/m_num));
            }
            if(lpi>2*PI/2 && lpi<=2*PI*3/4) //大于180度小于等于270度
            {
                arrays[i] = CPoint(x-r*sin(2*i*PI/m_num-2*PI/2),r+r*cos(2*i*PI/m_num-2*PI/2));
            }
            if(lpi>2*PI*3/4 && lpi<=2*PI) //大于270度小于等于360度
            {
                arrays[i] = CPoint(x-r*sin(2*PI-2*i*PI/m_num),r-r*cos(2*PI-2*i*PI/m_num));
            }
        }
    }
    dc.SetBkMode(TRANSPARENT); //设置背景透明
    CBrush brush(m_color); //创建一个位图画刷
    dc.SelectObject(&brush);
    CPen pen(PS_NULL,1,m_color); //创建画笔
    dc.SelectObject(&pen);
    if(m_result)
    {
        dc.Polygon(arrays,m_num); //绘制多边形
    }
    else
    {
        dc.Ellipse(0,0,rect.Width(),rect.Height()); //绘制圆形
    }
    if(IsPressed) //判断鼠标是否按下
    {
        CPen pen(PS_DASHDOTDOT,2,RGB(0,0,0)); //创建画笔
        dc.SelectObject(&pen);
        if(m_result)
        {
            dc.MoveTo(arrays[0]); //设置起点
            for(int i=1;i<m_num;i++)
            {
                dc.LineTo(arrays[i]); //画线
            }
            dc.LineTo(arrays[0]);
        }
        else
        {
            dc.Ellipse(0,0,rect.Width(),rect.Height()); //绘制圆形
        }
    }
    else
    {
        CPen pen(PS_DASHDOTDOT,2,m_color); //设置画笔
        dc.SelectObject(&pen);
        if(m_result)
        {
            dc.MoveTo(arrays[0]); //设置顶点
            for(int i=1;i<m_num;i++)
            {
                dc.LineTo(arrays[i]); //画多边形边线
            }
            dc.LineTo(arrays[0]);
        }
    }
}
```

```

    }
    else
        dc.Ellipse(0,0,rect.Width(),rect.Height());    //绘制圆形
    }

    CString str;
    GetWindowText(str);
    dc.SetTextColor(0,0,0);    //获得按钮文本
    //设置文本颜色
    //绘制按钮文本
    dc.DrawText(str,CRect(0,0,rect.right,rect.bottom),DT_CENTER|DT_VCENTER|DT_SINGLELINE);
}

```

举一反三

根据本实例，读者可以：

- 创建其他形状按钮。

实例 053 热点按钮

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\02\053

实例说明

通常情况下，人们的眼睛会追随动态的东西，所以当鼠标滑过热点按钮时，按钮发生变化，自然就可以引起用户的注意了，热点按钮可以起到这样的效果。运行程序，热点按钮效果如图 2.5 所示。

技术要点

要实现具有热点效果的按钮控件，主要用到 GetCursorPos 方法和 PtInRect 方法。

- (1) GetCursorPos 方法。GetCursorPos 方法用于获得鼠标的当前坐标，语法如下：

```
BOOL GetCursorPos( LPPOINT lpPoint );
```

参数说明：

- lpPoint：指向鼠标当前位置的 POINT 结构指针。

- (2) PtInRect 方法。PtInRect 方法用于判断指定点是否在指定矩形区域内，语法如下：

```
BOOL PtInRect( POINT point ) const;
```

参数说明：

- point：包含一个 POINT 结构或 CPoint 对象。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 以 CButton 类为基类派生一个 CButtonHot 类。
- (3) 向工程导入 4 个 BMP 位图资源，并向对话框中添加 2 个按钮控件，为按钮控件选择 Owner Draw 属性。

- (4) 在 CButtonHot 类的头文件中声明变量，代码如下：

```

UINT m_DownPic;    //鼠标按下时显示的图片
UINT m_NomalPic;    //正常情况下显示的图片
UINT m_EnablePic;    //按钮失效时显示的图片
UINT m_MovePic;    //鼠标经过按钮时显示的图片
BOOL m_IsInRect;    //是否在按钮区域内

```

- (5) 在 CButtonHot 类的构造函数中初始化变量，代码如下：

```

CButtonHot::CButtonHot()
{
    m_DownPic = IDB_BUTTONDOWN;    //鼠标按下时显示的图片
    m_NomalPic = IDB_BUTTONUP;    //正常情况下显示的图片
}

```

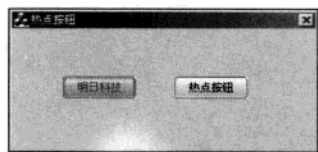


图 2.5 热点按钮

```
m_EnablePic = IDB_BUTTONENABLE; //按钮失效时显示的图片
m_MovePic = IDB_BUTTONMOVE; //鼠标经过按钮时显示的图片
m_IsInRect = FALSE;
```

(6) 在 CButtonHot 类中, 添加自定义函数 DrawBK, 该函数用于绘制按钮控件的背景位图, 代码如下:

```
void CButtonHot::DrawBK(CDC *pDC, UINT ResID)
{
    CDC memDC;
    memDC.CreateCompatibleDC(pDC); //创建兼容的设备上下文
    CRect rect; //声明区域对象
    GetClientRect(rect); //获得编辑框客户区域
    CBitmap bitmap;
    BITMAP bitStruct;
    bitmap.LoadBitmap(ResID); //加载位图资源
    bitmap.GetBitmap(&bitStruct); //获得位图资源信息
    memDC.SelectObject(&bitmap); //选入位图对象
    pDC->StretchBlt(0,0,rect.Width(),rect.Height(),&memDC,0,0,bitStruct.bmWidth,
        bitStruct.bmHeight,SRCCOPY); //绘制背景
    memDC.DeleteDC(); //删除设备上下文
    bitmap.DeleteObject(); //删除位图对象
}
```

(7) 在 CButtonHot 类中, 重载 DrawItem 虚函数, 在该虚函数中根据按钮状态绘制按钮的背景图片, 代码如下:

```
void CButtonHot::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
    CDC dc; //声明设备上下文
    dc.Attach(lpDrawItemStruct->hDC); //获得绘制按钮设备上下文
    UINT state = lpDrawItemStruct->itemState; //获取状态
    CRect rect; //声明区域对象
    GetClientRect(rect); //获得编辑框客户区域
    CString text; //声明字符串变量
    GetWindowText(text); //获得控件显示文本
    if(state & ODS_DISABLED) //如果不可用
    {
        DrawBK(&dc,m_EnablePic); //绘制不可用背景
        dc.SetTextColor(RGB(0,0,0)); //设置文本颜色
    }
    else if(state&ODS_SELECTED) //如果选择按钮
    {
        DrawBK(&dc,m_DownPic); //绘制选择状态背景
        dc.SetTextColor(RGB(0,0,255)); //设置文本颜色
    }
    else if(m_IsInRect==TRUE) //如果是热点
    {
        DrawBK(&dc,m_MovePic); //绘制热点状态背景
        dc.SetTextColor(RGB(255,0,0)); //绘制文本颜色
    }
    else //默认情况下
    {
        DrawBK(&dc,m_NomalPic); //绘制默认按钮状态背景
        dc.SetTextColor(RGB(0,0,0)); //绘制文本颜色
    }
    if(state&ODS_FOCUS) //如果获得焦点
    {
        CRect FocText(rect); //构造焦点区域
        FocText.DeflateRect(2,2,2,2); //设置焦点区域大小
        dc.DrawFocusRect(&FocText); //绘制焦点框
        lpDrawItemStruct->itemAction = ODA_FOCUS ;
    }
    dc.SetBkMode(TRANSPARENT); //设置背景透明
    dc.DrawText(text,&rect,DT_CENTER|DT_VCENTER|DT_SINGLELINE); //绘制按钮文本
}
```

(8) 在 CButtonHot 类中, 重载 PreSubclassWindow 虚函数, 在该虚函数中设置定时器, 代码如下:

```
void CButtonHot::PreSubclassWindow()
{
    SetTimer(1,10,NULL); //设置定时器
    CButton::PreSubclassWindow();
}
```


(9) 在 CButtonHot 类中, 处理按钮的 WM_TIMER 事件, 在该事件的处理函数中判断按钮是否为热点效果, 代码如下:

```
void CButtonHot::OnTimer(UINT nIDEvent)
{
    CPoint point;                //声明Cpoint变量
    GetCursorPos(&point);        //获得鼠标光标位置
    CRect rcWnd;                //声明区域对象
    GetWindowRect(&rcWnd);      //获得按钮区域
    if(rcWnd.PtInRect(point))    //判断鼠标光标是否在按钮上
    {
        if(m_IsInRect == TRUE)  //判断鼠标光标是否一直在按钮上
            goto END;          //跳转到标记
        else                    //鼠标光标移动到按钮上
        {
            m_IsInRect = TRUE;  //设置m_IsInRect变量值
            Invalidate();        //重绘按钮
        }
    }
    else                        //不在按钮区域内
    {
        if(m_IsInRect == FALSE) //判断鼠标光标一直在按钮外
            goto END;          //跳转到标记
        else                    //鼠标光标移动到按钮外
        {
            Invalidate();        //重绘按钮
            m_IsInRect = FALSE;  //设置m_IsInRect变量值
        }
    }
    END: CButton::OnTimer(nIDEvent); //设置标记, 调用基类方法
}
```

(10) 在 CButtonHot 类中, 处理按钮的 WM_ERASEBKGD 事件, 在该事件的处理函数中禁止调用基类方法重绘按钮控件, 代码如下:

```
BOOL CButtonHot::OnEraseBkgnd(CDC* pDC)
{
    return true; //CButton::OnEraseBkgnd(pDC); //不调用基类方法
}
```

(11) 在 CButtonHot 类中, 重载 PreTranslateMessage 虚函数, 在该函数中截获回车键按下和释放事件, 并修改为鼠标左键的按下和释放事件, 代码如下:

```
BOOL CButtonHot::PreTranslateMessage(MSG* pMsg)
{
    //截获回车键的按下事件
    if(pMsg->hwnd==this->GetSafeHwnd()&&pMsg->message==WM_KEYDOWN && pMsg->wParam==13)
    {
        pMsg->lParam = 0;
        pMsg->message = WM_LBUTTONDOWN; //改为鼠标左键的按下事件
    }
    //截获回车键的释放事件
    if(pMsg->hwnd==this->GetSafeHwnd()&&pMsg->message==WM_KEYUP && pMsg->wParam==13)
    {
        pMsg->lParam = 0;
        pMsg->message = WM_LBUTTONUP; //改为鼠标左键的释放事件
    }
    return CButton::PreTranslateMessage(pMsg);
}
```

举一反三

根据本实例, 读者可以:

- 根据按钮状态显示不同的位图。

2.2 编辑框控件典型实例

编辑框控件主要用来显示、获取、编辑和修改文本信息, 是程序与用户交流的一个主要控件。在程序运行中, 具有良好的交互性。本节主要介绍应用编辑框控件的典型实例。

实例 054

为编辑框设置新的系统菜单

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\02\054

实例说明

默认情况下编辑框控件的菜单是针对文本的菜单命令：剪切、复制和粘贴等，本实例实现将编辑框控件的菜单改为用户自定义菜单，程序运行结果如图 2.6 所示。

技术要点

本实例实现覆写 PreTranslateMessage 函数，实现在 Text Box 控件窗体上对 WM_RBUTTONDOWN 消息的处理，当程序产生 WM_RBUTTONDOWN 消息时调用 TrackPopupMenu 函数来显示右键菜单，它的语法如下：

```
BOOL TrackPopupMenu( UINT nFlags, int x, int y, CWnd* pWnd, LPCRECT lpRect = NULL );
```

参数说明：

- nFlags：鼠标按钮标识和屏幕位置标识，取值如下。
 - TPM_CENTERALIGN：根据 x 坐标水平居中。
 - TPM_LEFTALIGN：根据 x 坐标左对齐。
 - TPM_RIGHTALIGN：根据 x 坐标右对齐。
 - TPM_LEFTBUTTON：左键菜单。
 - TPM_RIGHTBUTTON：右键菜单。
- x：菜单左顶点 x 轴坐标。
- y：菜单左顶点 y 轴坐标。
- pWnd：菜单显示的窗体指针。
- lpRect：窗体矩形指针。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框上添加一个静态文本控件，设置 Caption 属性为“通过右键菜单可以对编辑框中字符进行特殊操作”；添加一个编辑框控件，设置 ID 属性为 IDC_EDTEXT。
- (3) 在工程中添加 Menu 资源，设置 ID 属性为 IDR_TEXTMENU。
- (4) 在 TextNewMenuDlg.h 文件中加入变量声明：

```
CMenu menu;
```

- (5) 在 OnInitDialog 中装载菜单，代码如下：

```
BOOL CTextNewMenuDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    ...//此处代码省略
    menu.LoadMenu(IDR_TEXTMENU);           //加载菜单资源
    return TRUE;
}
```

- (6) 通过 PreTranslateMessage 对鼠标右键进行处理，代码如下：

```
BOOL CTextNewMenuDlg::PreTranslateMessage(MSG* pMsg)
{
    if(pMsg->message==WM_RBUTTONDOWN && pMsg->hwnd== m_mytext.m_hWnd)//在编辑框中释放鼠标右键
    {
        CMenu* pPopup = menu.GetSubMenu(0);           //获得子菜单
        CRect rc;
        CPoint point=pMsg->pt;
        rc.top=point.x;
```

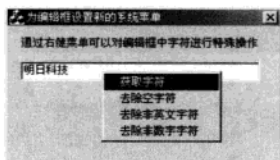


图 2.6 为编辑框设置新的系统菜单

```

rc.left=point.y;
pPopup->TrackPopupMenu(TPM_LEFTALIGN|TPM_LEFTBUTTON|TPM_VERTICAL,rc.top
,rc.left,this,&rc);           //显示弹出菜单
}
return CDialog::PreTranslateMessage(pMsg);
}

```

(7) 右键菜单项的单击事件处理函数代码如下:

```

void CTextNewMenuDlg::OnView1()
{
    AfxMessageBox("获取字符");
}
void CTextNewMenuDlg::OnView2()
{
    AfxMessageBox("去除空字符");
}
void CTextNewMenuDlg::OnView3()
{
    AfxMessageBox("去除非英文字符");
}
void CTextNewMenuDlg::OnView4()
{
    AfxMessageBox("去除非数字字符");
}

```

举一反三

根据本实例,读者可以:

- 实现列表控件的右键菜单。

实例 055

为编辑框控件添加列表选择框

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\02\055

实例说明

编辑框控件需要用户输入文本,但是反复输入相同的内容是很麻烦的。本实例实现在用户输入文本时,程序自动在数据库中查询,如果查到有类似的信息后,则以列表形式显示在编辑框控件下,用户可以根据需要选择列表中的内容。程序运行结果如图2.7所示。

技术要点

本实例的实现需要覆写 `PreTranslateMessage` 函数以及对编辑框控件的 `EN_CHANGE` 消息进行处理。`EN_CHANGE` 消息在编辑框控件中的文本内容发生变化时产生,用户每输入一个字符文本内容都会产生 `EN_CHANGE` 消息,处理 `EN_CHANGE` 消息主要是在数据库中查找编辑框控件中的文本内容。覆写 `PreTranslateMessage` 函数主要是在出现列表控件以后,处理用户按上下方向键及鼠标单击列表控件时产生的消息。

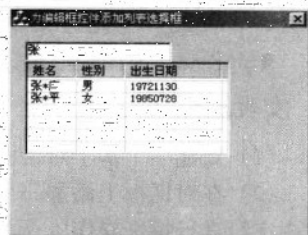


图 2.7 为编辑框控件添加列表选择框

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框上添加一个编辑框控件,设置 ID 属性为 `IDC_EDOBJ`,添加成员变量 `m_edobj`,添加一个列表视图控件,设置 ID 属性为 `IDC_TIPLIST`,添加成员变量 `m_tiplist`。
- (3) 为编辑框控件添加 `EN_CHANGE` 消息处理函数 `OnChangeEdobj`;为列表视图控件添加 `NM_DBLCLK` 消息处理函数 `OnDbclclkTiplist`。
- (4) 在 `TextboxListDlg.h` 文件中添加变量声明:

```

CString xm,xb,csrq,gzdw,yddh,gddh;
bool IsShowing;

```

(5) 通过 PreTranslateMessage 对键盘按键进行处理, 代码如下:

```
BOOL CTextboxListDlg::PreTranslateMessage(MSG* pMsg)
{
    if(pMsg->message==WM_KEYDOWN && pMsg->wParam==VK_ESCAPE)//按下<ESC>键
    {
        m_tiplist.ShowWindow(SW_HIDE);           //不显示提示列表
        IsShowing=false;
        pMsg->wParam=VK_CONTROL;
    }
    if (pMsg->message==WM_LBUTTONDOWN)           //按下鼠标左键
    {
        if (pMsg->hwnd!= m_tiplist.m_hWnd)       //当前窗口不是列表视图控件
        {
            m_tiplist.ShowWindow(SW_HIDE);       //隐藏提示列表
            IsShowing=false;
        }
    }
    if(pMsg->message==WM_KEYDOWN && pMsg->wParam==13) //按下回车键
    {
        if(IsShowing)
        {
            m_edobj.SetWindowText(xm);           //设置编辑框显示数据
            m_tiplist.ShowWindow(SW_HIDE);       //隐藏提示列表
            IsShowing=false;
            i=0;
            pMsg->wParam=VK_CONTROL;
        }
    }
    if(pMsg->hwnd==m_tiplist.m_hWnd&& pMsg->message==WM_LBUTTONDBLCLK) //在提示列表中双击鼠标左键
    {
        m_edobj.SetWindowText(xm);           //设置编辑框显示数据
        m_tiplist.ShowWindow(SW_HIDE);       //隐藏提示列表
        IsShowing=false;
    }
    if(pMsg->message==WM_KEYDOWN && pMsg->wParam==VK_DOWN) //按下下箭头
    {
        if(IsShowing)
        {
            if(i==m_tiplist.GetItemCount())       //获得列表记录数
            {
                i=0;
                m_tiplist.SetHotItem(i);
                xm=m_tiplist.GetItemText(i,0);    //获得列表项数据
                i+=1;
            }
        }
    }
    return CDialog::PreTranslateMessage(pMsg);
}
```

(6) 自定义函数 AutoPostion, 实现提示窗体位置的调整, 代码如下:

```
void CTextboxListDlg::AutoPostion()
{
    this->m_tiplist.MoveWindow(15,36,210,100);
}
```

(7) 编辑框 IDC_EDOBJ 的 EN_CHANGE 消息的实现函数的代码如下:

```
void CTextboxListDlg::OnChangeEdobj()
{
    CString edit;
    m_edobj.GetWindowText(edit);               //获得编辑框中数据
    if(!edit.IsEmpty())                       //如果数据不为空
    {
        this->AutoPostion();                  //设计提示列表位置
        m_tiplist.DeleteAllItems();           //删除提示列表中数据
        this->SetDataBase(edit);               //在数据库中查找编辑框中数据
        if(m_tiplist.GetItemCount()>0)       //如果列表项数量大于0
        {
            m_tiplist.ShowWindow(SW_SHOW);    //显示提示列表
            IsShowing=true;
        }
    }
    else
    {
        m_tiplist.ShowWindow(SW_HIDE);       //隐藏提示列表
        IsShowing=false;
    }
}
```

(8) 控件 IDC_TIPLIST 的 NM_DBLCLK 消息的函数的代码如下:


```
void CTextboxListDlg::OnDblclkTiplist(NMHDR* pNMHDR, LRESULT* pResult)
{
    int i=m_tiplist.GetHotItem();
    CString text;
    xm=m_tiplist.GetItemText(i,1);
    *pResult = 0;
} //获得列表项数据
```

举一反三

根据本实例，读者可以：

- 实现多个编辑框控件的列表提示。

实例 056 多彩边框的编辑框

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\02\056

实例说明

本实例改变了传统的编辑框风格。运行程序，五颜六色的编辑框将显示在窗体中，如图 2.8 所示。

技术要点

以 CEdit 类为基类派生 CcolourEdit 类，然后设置 WM_CTLCOLOR 消息，使用 FrameRect 函数重绘编辑框的边框。

FrameRect 函数：在矩形周围绘制边框。函数原型如下：

```
void FrameRect( LPCRECT lpRect, CBrush* pBrush );
```

参数说明：

- lpRect：对要描绘的边框进行描述的一个矩形。这等效于将画笔设置成一个单位的宽度，然后用矩形函数画出一个矩形。
- pBrush：欲使用的刷子的句柄。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加 8 个编辑框控件。
- (3) 通过 New Class 窗口生成一个新类 CcolourEdit，基类为 CEdit。添加一个 COLORREF 类型的成员变量 m_Colour。
- (4) 主要程序代码如下：

```
HBRUSH CColourEdit::CtlColor(CDC* pDC, UINT nCtlColor)
{
    CDC* dc = GetDC(); //获取画布对象
    CRect rect;
    GetClientRect(rect); //获取客户区域
    rect.InflateRect(1,1,1,1); //将客户区域增大一个像素
    CBrush brush( m_Colour); //创建画刷
    dc->FrameRect(rect,&brush); //绘制边框
    return NULL;
}
```

举一反三

根据本实例，读者可以：

- 设置文本框的边框颜色。

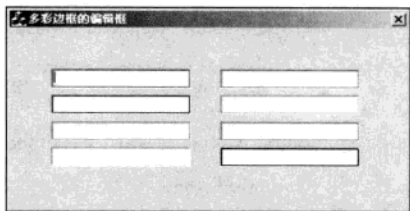


图 2.8 多彩边框的编辑框

实例 057 改变编辑框文本颜色

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\02\057

实例说明

在实际的应用程序中，除了改变编辑框的背景和边框颜色外，还可以改变编辑框中字体的颜色。运行程序，在编辑框中输入文本，在单选按钮组中选择文本的颜色，运行结果如图 2.9 所示。

技术要点

可以使用 SetTextColor 函数改变编辑框中文本的颜色，通过对消息 WM_CTLCOLOR 的响应函数来实现对 SetTextColor 函数的调用。在控件被显示之前，WM_CTLCOLOR 消息被发送到控件所在的对话框，这个消息的响应函数会修改对话框及其中控件的颜色。该消息的响应函数原型如下：

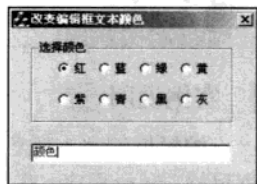


图 2.9 改变编辑框文本颜色

```
afx_msg HBRUSH OnCtlColor( CDC* pDC, CWnd* pWnd, UINT nCtlColor );
```

参数说明：

- pDC：指向绘图设备的指针。
- pWnd：指向具体控件的指针。
- nCtlColor：控件的类型，其值如表 2.1 所示。

表 2.1 nCtlColor 参数可选值表

参 数	描 述
CTLCOLOR_BTN	按钮类控件
CTLCOLOR_DLG	对话框
CTLCOLOR_EDIT	编辑框控件
CTLCOLOR_LISTBOX	列表框控件
CTLCOLOR_MSGBOX	消息框控件
CTLCOLOR_SCROLLBAR	滚动条控件
CTLCOLOR_STATIC	静态文本控件

SetTextColor 函数的原型如下：

```
virtual COLORREF SetTextColor( COLORREF crColor );
```

参数说明：

- crColor：要设置的字体颜色。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加 1 个编辑框控件和 8 个单选按钮控件。
- (3) 主要程序代码如下：

```
HBRUSH CColourTextDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
    if(nCtlColor == CTLCOLOR_EDIT) //是编辑框
        pDC->SetTextColor(colour); //设置文本颜色
    return hbr;
}
```

举一反三

根据本实例，读者可以：

- 设置其他控件的字体颜色；
- 设置控件的背景颜色。

实例 058 不同文本颜色的编辑框

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\02\058

实例说明

编辑框的文本颜色常用的是黑色，白色的背景衬托黑色的字体，本实例改变了传统的编辑框风格。运行程序，在编辑框中显示的文字将具有不同的文本颜色，如图 2.10 所示。

技术要点

在设置文本颜色时需要使用 `CreateStockObject` 函数，该函数获取预定义的 Windows GDI 的画笔、画刷和字体句柄，并将 GDI 对象与 `CGdiObject` 类对象相关联。语法如下：

```
BOOL CreateStockObject( int nIndex );
```

参数说明：

- `nIndex`：定义标准对象类型的常量，可选值如下。
 - `BLACK_BRUSH`：黑色刷子。
 - `DKGRAY_BRUSH`：黑灰色刷子。
 - `GRAY_BRUSH`：灰色刷子。
 - `HOLLOW_BRUSH`：凹刷子。
 - `LTGRAY_BRUSH`：浅灰色刷子。
 - `NULL_BRUSH`：空刷子。
 - `WHITE_BRUSH`：白色刷子。
 - `BLACK_PEN`：黑色画笔。
 - `WHITE_PEN`：白色画笔。
 - `ANSI_FIXED_FONT`：采用 Windows (ANSI) 字符集的等宽字体。
 - `ANSI_VAR_FONT`：采用 Windows (ANSI) 字符集的不等宽字体。
 - `DEVICE_DEFAULT_FONT`：设备使用的默认字体 (NT)。
 - `DEFAULT_GUI_FONT`：用户界面的默认字体，包括菜单和对话框字体。
 - `OEM_FIXED_FONT`：OEM 字符集的固有字体。
 - `SYSTEM_FONT`：屏幕系统字体。这是用于菜单、对话框等的默认不等宽字体。
 - `SYSTEM_FIXED_FONT`：屏幕系统字体。这是用于菜单、对话框等的默认等宽字体。
 - `DEFAULT_PALETTE`：默认调色板。

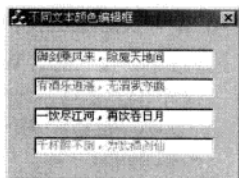


图 2.10 不同文本颜色的编辑框

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 创建一个以 `CEdit` 类为基类的派生类 `CColorEdit`。
- (3) 在 `CColorEdit` 类的头文件中声明一个 `COLORREF` 类型变量 `m_Color`。

(4) 向对话框中添加6个编辑框控件,通过类向导为控件关联 CColorEdit 类成员变量。

(5) 手动添加一个 SetColor 函数,用来设置文本颜色的变量赋值,代码如下:

```
void CColorEdit::SetColor(COLORREF color)           //SetColor函数
{
    m_Color = color;                               //为变量m_Color赋值
}
```

(6) 处理 CColorEdit 类的 WM_CTLCOLOR 消息,在该消息的处理函数中设置文本颜色,代码如下:

```
HBRUSH CColorEdit::CtlColor(CDC* pDC, UINT nCtlColor) //WM_CTLCOLOR消息处理函数
{
    CBrush m_Brush;                                  //声明画刷对象
    m_Brush.CreateStockObject(WHITE_BRUSH);          //创建画刷
    pDC->SetTextColor(m_Color);                      //设置文本颜色
    return m_Brush;                                  //返回画刷
}
```

(7) 在对话框的 OnInitDialog 函数中为编辑框控件设置文本显示颜色,代码如下:

```
m_Edit1.SetColor(RGB(255,0,0));
m_Edit2.SetColor(RGB(0,0,255));
m_Edit3.SetColor(RGB(255,0,255));
m_Edit4.SetColor(RGB(0,255,0));
```

举一反三

根据本实例,读者可以:

- 设置编辑框控件显示不同字体。

实例 059 位图背景编辑框

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\02\059

实例说明

在实际的应用程序中,白色背景的编辑框让人看起来觉得乏味,为了更好地美化程序,从而吸引用户,可以设置位图背景编辑框。运行程序,在编辑框中输入文本,运行结果如图 2.11 所示。

技术要点

首先使用 SetBkMode 函数设置编辑框中文本背景透明,然后在 WM_ERASEBKGD 消息的响应函数中来实现对编辑框背景的绘制。SetBkMode 函数的语法如下:

```
int SetBkMode( int nBkMode );
```

参数说明:

- nBkMode: 指向绘图设备的指针。
 - OPAQUE: 缺省背景模式。
 - TRANSPARENT: 背景在绘图之前不改变。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 创建一个以 CEdit 类为基类的派生类 CBmpEdit。
- (3) 选择工作区窗口的 ResourceView 选项卡,向对话框中导入一个位图资源。
- (4) 在 CBmpEdit 类的头文件中声明一个 CBitmap 类对象 m_Bitmap。
- (5) 在 CBmpEdit 类的构造函数中加载位图资源,代码如下:

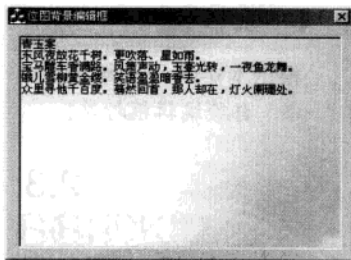


图 2.11 位图背景编辑框


```
m_Bitmap.LoadBitmap(IDB_BITMAP1);
```

```
//加载位图资源
```

(6) 处理 CBmpEdit 类的 WM_CTLCOLOR 消息, 在该消息的处理函数中设置文本的背景透明, 代码如下:

```
HBRUSH CBmpEdit::CtlColor(CDC* pDC, UINT nCtlColor)    //WM_CTLCOLOR消息处理函数
{
    pDC->SetBkMode(TRANSPARENT);                        //设置文本背景透明
    return NULL;
}
```

(7) 处理 CBmpEdit 类的 WM_ERASEBKGD 消息, 在该消息的处理函数中绘制编辑框背景, 代码如下:

```
BOOL CBmpEdit::OnEraseBkgnd(CDC* pDC)                //消息处理函数
{
    CDC memDC;                                          //设备上下文
    memDC.CreateCompatibleDC(pDC);                    //创建内存设备上下文
    memDC.SelectObject(&m_Bitmap);                     //将位图选入设备上下文
    BITMAP m_Bmp;                                       //声明BITMAP对象
    m_Bitmap.GetBitmap(&m_Bmp);                         //获得位图信息
    int x = m_Bmp.bmWidth;                             //获得位图的宽度
    int y = m_Bmp.bmHeight;                             //获得位图的高度
    CRect rect;                                         //声明区域对象
    GetClientRect(rect);                               //获得编辑框客户区域
    pDC->StretchBlt(0,0,rect.Width(),rect.Height(),&memDC,0,0,x,y,SRCCOPY); //绘制位图背景
    memDC.DeleteDC();                                   //释放内存设备上下文
    return TRUE;                                       //返回真值
    //return CEdit::OnEraseBkgnd(pDC);                 //禁止调用基类方法
}
```

(8) 处理 CBmpEdit 类的 EN_CHANGE 消息, 在该消息的处理函数中重绘背景, 代码如下:

```
void CBmpEdit::OnChange()                             //EN_CHANGE消息处理函数
{
    Invalidate();                                       //重绘背景
}
```

举一反三

根据本实例, 读者可以:

- 自绘编辑框控件。

2.3 静态文本控件典型实例

静态文本控件 (Static Text) 是一种单向交互的控件, 用于显示数据, 但是不接受输入, 本节通过几个实例介绍静态文本控件在程序中的应用。

实例 060 电子时钟

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\02\060

实例说明

从出生开始, 一起伴随着人们成长的东西不多, 时间就是其中一样, 在日常的生活中任何人都离不开时间, 而时钟则是人们判断时间的重要依据, 而在许多地方, 悬挂的已经不在是老式的石英钟, 而是电子时钟。这种数字的时间显示可以让人们更直观的判断时间, 那么如何在程序中实现电子时钟的数字变化呢? 本实例就来为读者解决这一问题, 程序运行效果如图 2.12 所示。

技术要点

在设计迷你电子时钟控件时，主要是根据位图来绘制数字。首先真准备一副位图，其中存储了0~9共10个数字和两个默认效果的图片，如图2.13所示。



图 2.12 电子时钟



图 2.13 时钟数字位图

在静态文本控件中，根据当前显示的数字，从位图中查找对应的图片（位图中每一个图片的大小是相同的），将其绘制在静态文本控件的窗口中，这样就实现了电子时钟形式的数字。在设计电子时钟时，由于需要不停的在静态文本控件中绘制位图，为了放置出现界面闪烁，笔者定义了一个内存画布，对静态本控件的绘图操作在内存画布上进行，在内存画布释放时将其内容输出到静态文本控件中。下面给出定义内存画布的相关代码。

```
class CMemDC : public CDC
{
private:
    CBitmap*      m_bitmap;           //定义位图对象指针
    CBitmap*      m_oldbmp;
    CDC*          m_pDC;              //定义源设备上下文
    CRect         m_Rect;             //定义源设备上下文区域大小
public:
    CMemDC(CDC* pDC, const CRect& rect) : CDC() //构造函数
    {
        CreateCompatibleDC(pDC);           //创建兼容的设备上下文
        m_bitmap = new CBitmap;             //创建位图对象
        m_bitmap->CreateCompatibleBitmap(pDC, rect.Width(), rect.Height()); //创建兼容的位图
        m_oldbmp = SelectObject(m_bitmap);  //选中位图对象
        m_pDC = pDC;                        //记录源设备上下文
        m_Rect = rect;                      //记录源设备上下文区域
    }
    ~CMemDC()                               //构造函数
    {
        //将内存画布中的内容复制到源设备上下文中
        m_pDC->BitBlt(m_Rect.left, m_Rect.top, m_Rect.Width(), m_Rect.Height(),
                     this, m_Rect.left, m_Rect.top, SRCCOPY);
        SelectObject(m_oldbmp);
        if (m_bitmap != NULL)
            delete m_bitmap;                //释放位图对象
    }
};
```

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 单击“Insert”/“Resource”菜单项，在打开的 Insert Resource 对话框中单击 Import 按钮，向工程中导入一个位图资源。
- (3) 向窗体中添加一个图片控件和一个静态文本控件，右键单击图片控件，在弹出的菜单中选择 Properties 选项，设置 Type 属性为 Bitmap，设置 Image 属性为 IDB_BITMAP1。
- (4) 以 CStatic 类为基类派生一个 CNumberCtrl 类，并为要自绘的静态文本控件关联一个该类的对象。
- (5) 在 CNumberCtrl 类的 OnPaint 方法中绘制控件外观，代码如下：

```
void CNumberCtrl::OnPaint()
{
```

```

CPaintDC dc(this);
SetRedraw(FALSE); //禁止窗口绘制
CRect clientRC;
GetClientRect(clientRC); //获取窗口客户区域
CMemDC memDC(&dc, clientRC); //定义内存画布
SetWindowText("");
CBitmap bmp;
bmp.LoadBitmap(IDB_NUMBERBMP); //加载位图
CDC tmpDC;
tmpDC.CreateCompatibleDC(&dc); //创建一个兼容的设备上下文
tmpDC.SelectObject(&bmp); //选中位图对象
BITMAP bInfo;
bmp.GetBitmap(&bInfo); //定义位图信息
int nbmpWidth = bInfo.bmWidth; //获取位图信息
int nbmpHeight = bInfo.bmHeight; //获取位图高度
int nLen = m_csText.GetLength(); //获取文本长度
for (int i=0; i<m_nNumberLen; i++) //绘制背景
{
    memDC.BitBlt((i)*m_nNumberWidth, 0, m_nNumberWidth, nbmpHeight,
        &tmpDC, 10*m_nNumberWidth, 0, SRCCOPY);
}
if (nLen>0 && nLen<=m_nNumberLen) //判断数字是否合法
{
    for (int n=0; n<nLen; n++)
    {
        char ch = m_csText[nLen-n-1];
        if (ch == ':')
        {
            memDC.BitBlt((m_nNumberLen-10)*m_nNumberWidth, 0, m_nNumberWidth,
                nbmpHeight, &tmpDC, m_nNumberWidth, 0, SRCCOPY);
        }
        else
        {
            int nCh = atoi(&ch);
            //绘制数字位图
            memDC.BitBlt((m_nNumberLen-n-1)*m_nNumberWidth, 0, m_nNumberWidth,
                nbmpHeight, &tmpDC, (nCh)*m_nNumberWidth, 0, SRCCOPY);
        }
    }
}
bmp.DeleteObject(); //删除位图对象
tmpDC.DeleteDC();
SetRedraw(); //激活窗口绘制
}

```

举一反三

根据本实例，读者可以：

- 设计倒计时器。

实例 061 文本背景的透明处理

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\02\061

实例说明

在设计应用程序界面时，有时需要使静态文本控件背景透明。例如，将静态文本控件放置在图片上，如果静态文本控件显示灰色的背景，将影响界面美观。如何将静态文本控件背景透明呢？本例实现了这一功能，效果如图 2.14 所示。

技术要点

通常情况下，可以通过自绘静态文本控件的方法，在其 WM_PAINT 消息处理函数中重新绘制文本并设置背景透明。但是这样做比较麻烦，其实用户可以通过属性的设置来实现静态文本控件背景的透明。

- (1) 首先打开静态文本控件的属性窗口，并勾选 Simple 属性。

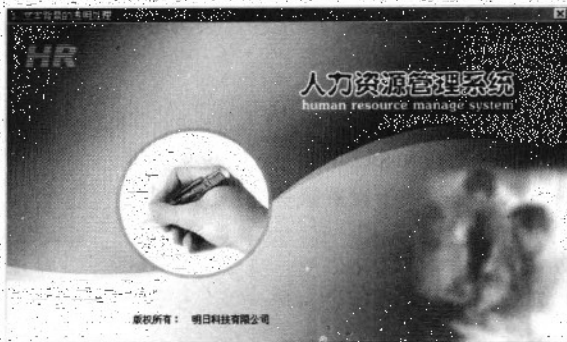


图2.14 文本背景的透明处理

(2) 然后处理对话框的 WM_CTLCOLOR 消息，在其消息处理函数中判断当前对象是否是静态文本控件，如果是，则执行调用 SetBkMode 方法设置文本背景透明。SetBkMode 方法的语法如下：

```
int SetBkMode(int nBkMode);
```

参数说明：

- nBkMode：指定要设置的模式，其参数值为 OPAQUE 时缺省背景模式。背景在文本、阴影画刷、笔绘制之前用当前背景色填充；当参数值为 TRANSPARENT 时，背景在绘图之前不改变，通过该值可以设置背景透明。

实现过程

(1) 新建一个基于对话框的应用程序。

(2) 单击“Insert”/“Resource”菜单项，在打开的 Insert Resource 对话框中单击 Import 按钮，向工程中导入一个位图资源。

(3) 向窗体中添加一个图片控件和一个静态文本控件，右键单击图片控件，在弹出的菜单中选择 Properties 选项，设置 Type 属性为 Bitmap，设置 Image 属性为 IDB_BITMAP1。

(4) 处理对话框的 WM_CTLCOLOR 消息，在其消息处理函数中判断当前对象是否是静态文本控件，如果是，则执行调用 SetBkMode 方法设置文本背景透明，代码如下：

```
HBRUSH CTransStaticDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
    if (nCtlColor == CTLCOLOR_STATIC)           //判断是否为静态文本控件
    {
        pDC->SetBkMode(TRANSPARENT);           //设置文本背景透明
    }
    return hbr;
}
```

举一反三

根据本实例，读者可以：

- 设置不同文本背景颜色。

实例 062 制作超链接控件

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\02\062

实例说明

设计应用程序时，为了让用户方便地通过程序访问某个网站，需要采用具有超级链接功能的控

件, 本实例通过静态文本控件设计一个超级链接控件, 实例运行结果如图 2.15 所示。

技术要点

本实例通过新建 SetCursor 函数和 ShellExecute 函数来实现控件的超链接功能。SetCursor 函数用于设置鼠标光标的样式, ShellExecute 函数用来打开超链接。

(1) SetCursor 函数, 语法如下:

```
HCURSOR SetCursor (HCURSOR hCursor);
```

参数说明:

● hCursor: 光标的句柄。

(2) ShellExecute 函数, 语法如下:

```
HINSTANCE APIENTRY ShellExecute(HWND hwnd,LPCTSTR lpOperation,LPCTSTR lpFile,LPCTSTR lpParameters,LPCTSTR lpDirectory,INT nShowCmd);
```

参数说明:

● hwnd: 窗口句柄。

● lpOperation: 执行的操作, 包括 open、print 和 explore。

● lpFile: 文件路径。

● lpParameters: 执行操作的参数。

● lpDirectory: 指定默认目录。

● nShowCmd: 是否显示。

实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在工程中添加一个以 CStatic 类为基类的类 CSuperLabel。

(3) 在对话框上添加 5 个静态文本控件和 3 个编辑框控件, 设置其中 1 个静态文本控件的 ID 为 IDC_CONNECT。

(4) 在 CSuperLabel 类的 OnPaint 方法中绘制控件外观, 代码如下:

```
void CSuperLabel::OnPaint()
{
    CPaintDC dc(this);
    CDC* pDC = GetDC();           //获得设备上下文
    CString text;
    GetWindowText(text);           //获得控件显示文本
    if (m_ConnectStr.IsEmpty())
        m_ConnectStr = text;       //设置超链接文本
    pDC->SetBkMode(TRANSPARENT);  //设置背景透明
    pDC->SetTextColor(RGB(0,0,255)); //设置文本颜色为蓝色
    pDC->SelectObject(&m_Font);     //选入字体对象
    pDC->TextOut(0,0,text);         //绘制超链接文本
}
```

(5) 重载 CSuperLabel 类的 PreSubclassWindow 虚方法, 在该虚方法中设置超链接文本和字体, 代码如下:

```
void CSuperLabel::PreSubclassWindow()
{
    GetWindowText(m_ConnectStr); //获得超链接文本
    CFont* pFont = GetFont();     //获得控件字体
    pFont->GetLogFont(&lfont);
    lfont.lfUnderline = TRUE;      //设置下划线
    m_Font.CreateFontIndirect(&lfont); //创建新字体
    CStatic::PreSubclassWindow();
}
```

(6) 处理 CSuperLabel 类的鼠标移动消息, 在该消息的处理函数中修改鼠标的样式, 代码如下:

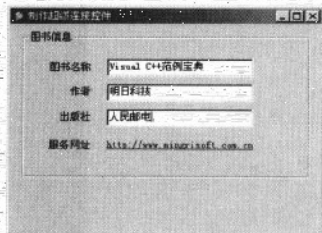


图 2.15 实现超级链接

```
void CSuperLabel::OnMouseMove(UINT nFlags, CPoint point)
{
    ::SetCursor(AfxGetApp()->LoadCursor(IDC_CURSOR1));           //设置鼠标样式
    CStatic::OnMouseMove(nFlags, point);
}
```

(7) 处理 CSuperLabel 类的鼠标左键按下消息, 在该消息的处理函数中打开超级链接, 代码如下:

```
void CSuperLabel::OnLButtonDown(UINT nFlags, CPoint point)
{
    ShellExecute(m_hWnd, NULL, m_ConnectStr, NULL, NULL, SW_SHOW); //打开超链接
    CStatic::OnLButtonDown(nFlags, point);
}
```

举一反三

根据本实例, 读者可以:

- 自绘静态文本控件。

2.4 列表框控件典型实例

利用列表框控件浏览数据具有简洁、直观的特点, 在开发应用程序时经常用到该控件, 本节通过几个实例介绍列表框控件在程序中的应用。

实例 063

利用列表框控件实现标签式数据选择

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\02\063

实例说明

本实例实现了列表控件的标签复选功能。运行程序, 列表中有 4 项, 分别对应窗体上面的 4 个按钮, 列表中哪一项处于选中状态, 对应的按钮就可用, 实例运行结果如图 2.16 所示。

技术要点

本实例主要通过 CCheckListBox 类实现, CCheckListBox 类是对 CListBox 类的扩充, 使 ListBox 具有标签复选功能。通过 CCheckListBox 类的 SetCheckStyle 方法实现列表的标签复选样式, 然后通过 SetCheck 方法使列表项前的复选框处于选中状态, 通过 GetCheck 方法获得列表项前的复选框状态, 如果方法返回 1 表示已经选中, 如果返回 0 表示没有被选中。

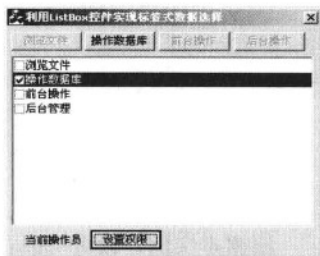


图 2.16 利用 ListBox 控件实现标签式数据选择

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框上添加 5 个按钮控件和 1 个列表框控件, 设置 ID 属性为 IDC_DATA LIST, 将 Owner draw 属性改为 Fixed, 并设置 Has strings 属性, 去除 Sort 属性。添加成员变量 m_chklist, 并将其改为继承自 CCheckListBox 类。
- (3) 在对话框初始化时向列表框中添加数据, 设置按钮控件不可用, 代码如下:

```
BOOL CListboxChkSelDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    //...此处代码省略
    m_chklist.SetCheckStyle(BS_CHECKBOX);
}
```



```
m_chklist.AddString("浏览文件");  
m_chklist.AddString("操作数据库");  
m_chklist.AddString("前台操作");  
m_chklist.AddString("后台管理");  
GetDlgItem(IDC_VIEW)->EnableWindow(FALSE);  
GetDlgItem(IDC_DATABASE)->EnableWindow(FALSE);  
GetDlgItem(IDC_FRONT)->EnableWindow(FALSE);  
GetDlgItem(IDC_BACK)->EnableWindow(FALSE);  
return TRUE;  
}
```

(4) 处理“设置权限”按钮的单击事件, 在该事件的处理函数中设置与列表中选中复选框的按钮可用, 代码如下:

```
void CListBoxChkSelDlg::OnOper()  
{  
    //判断列表项前的复选框是否选中, 如果选中则设置对应按钮可用  
    if(m_chklist.GetCheck(0))  
        GetDlgItem(IDC_VIEW)->EnableWindow(TRUE);  
    else  
        GetDlgItem(IDC_VIEW)->EnableWindow(FALSE);  
    if(m_chklist.GetCheck(1))  
        GetDlgItem(IDC_DATABASE)->EnableWindow(TRUE);  
    else  
        GetDlgItem(IDC_DATABASE)->EnableWindow(FALSE);  
    if(m_chklist.GetCheck(2))  
        GetDlgItem(IDC_FRONT)->EnableWindow(TRUE);  
    else  
        GetDlgItem(IDC_FRONT)->EnableWindow(FALSE);  
    if(m_chklist.GetCheck(3))  
        GetDlgItem(IDC_BACK)->EnableWindow(TRUE);  
    else  
        GetDlgItem(IDC_BACK)->EnableWindow(FALSE);  
}
```

(5) 处理列表框控件的双击事件, 当双击列表控件中的列表项时, 将对应列表项前的复选框选中, 代码如下:

```
void CListBoxChkSelDlg::OnDblickChklist()  
{  
    int i=m_chklist.GetCurSel();    //获得当前选中列表项索引  
    if(i<0)return;  
    if(m_chklist.GetCheck(i)<1)    //如果当前列表项没被勾选  
        m_chklist.SetCheck(i,1);    //勾选当前列表项的复选框  
    else                            //否则  
        m_chklist.SetCheck(i,0);    //去掉当前列表项的选择状态  
}
```

举一反三

根据本实例, 读者可以:

- 开发设置操作员权限的程序。

实例 064 以报表显示图书信息

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\064

实例说明

列表视图控件除了可以显示大、小图标和列表以外, 还可以显示报表数据信息, 本实例就是以报表的形式显示图书信息, 实例运行效果如图 2.17 所示。

技术要点

本实例以报表风格介绍列表视图控件的使用, 在使用报表风格时, 首先调用

SetExtendedStyle 方法设置列表视图控件的扩展风格。然后调用 InsertColumn 方法向列表视图控件添加列。之后才可以插入数据，在插入数据时先调用 InsertItem 方法插入行，接着调用 SetItemText 方法向列表的每一列插入数据。

(1) SetExtendedStyle 方法

SetExtendedStyle 方法用于设置列表视图控件的扩展风格。语法如下：

```
DWORD SetExtendedStyle( DWORD dwNewStyle );
```

参数说明：

- dwNewStyle：用于标识列表视图控件的扩展风格。

返回值：函数调用前的扩展风格。

(2) InsertColumn 方法

InsertColumn 方法用于设置列表视图控件的扩展风格。语法如下：

```
int InsertColumn( int nCol, const LVCOLUMN* pColumn );
```

```
int InsertColumn( int nCol, LPCTSTR lpszColumnHeading, int nFormat = LVCFMT_LEFT, int nWidth = -1, int nSubItem = -1 );
```

参数说明：

- nCol：标识新列的索引。
- pColumn：LVCOLUMN 结构指针，该结构中包含了列的详细信息。
- lpszColumnHeading：标识列标题。
- nFormat：标识列的对齐方式。
- nWidth：标识列宽度。
- nSubItem：标识关联当前列的子视图项索引。

(3) SetItemText 方法

SetItemText 方法用于向列表的每一列插入数据。语法如下：

```
BOOL SetItemText( int nItem, int nSubItem, LPCTSTR lpszText );
```

参数说明：

- hItem：表示项目行索引。
- nSubItem：表示项目列索引。
- lpszText：设置指定行、指定列中的显示文本。

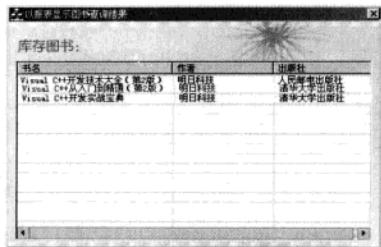


图 2.17 以报表显示图书查询结果

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加一个图片控件和一个列表视图控件。
- (3) 在对话框初始化时，设置列表视图控件的扩展风格，并设置列标题，向列表中插入图书信息。代码如下：

```
//设置列表视图的扩展风格
m_Grid.SetExtendedStyle(LVS_EX_FLATSB          //扁平风格显示滚动条
    LVS_EX_FULLROWSELECT                       //允许整行选中
    LVS_EX_HEADERDRAGDROP                     //允许整列拖动
    LVS_EX_ONECLICKACTIVATE                   //单击选中项
    LVS_EX_GRIDLINES);                        //画出网格线

//设置表头
m_Grid.InsertColumn(0,"书名",LVCFMT_LEFT,200,0); //设置书名列
m_Grid.InsertColumn(1,"作者",LVCFMT_LEFT,130,1); //设置作者列
m_Grid.InsertColumn(2,"出版社",LVCFMT_LEFT,130,2); //设置出版社列
m_Grid.InsertItem(0,"Visual C++开发技术大全 (第2版)"); //插入第0行
m_Grid.SetItemText(0,1,"明日科技"); //向第1列插入数据
m_Grid.SetItemText(0,2,"人民邮电出版社"); //向第2列插入数据
m_Grid.InsertItem(1,"Visual C++从入门到精通 (第2版)"); //插入第1行
m_Grid.SetItemText(1,1,"明日科技"); //向第1列插入数据
m_Grid.SetItemText(1,2,"清华大学出版社"); //向第2列插入数据
```



```

m_Grid.InsertItem(2,"Visual C++开发实战宝典");    //插入第2行
m_Grid.SetItemText(2,1,"明日科技");              //向第1列插入数据
m_Grid.SetItemText(2,2,"清华大学出版社");         //向第2列插入数据

```

举一反三

根据本实例,读者可以:

- 实现列表框控件中的数据管理。

实例 065 QQ 抽屉界面

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\065

实例说明

在当今社会,使用 QQ 进行通信的人越来越多,QQ 的好友列表用起来既方便又美观,所以深受用户喜爱,可是在 Visual C++ 的开发环境中没有类似的控件,那要使用这样的控件该怎么办呢?本实例通过 Visual C++ 实现了这种类似 QQ 抽屉效果的列表视图控件。运行本实例,单击“好友列表”按钮,将该按钮置顶,并显示该按钮下的列表项,实例运行效果如图 2.18 所示。

技术要点

本示例中实现 QQ 抽屉效果的列表视图控件时,主要用到了 SetItemPosition 方法和 Arrange 方法,下面对这两个方法进行介绍。

(1) SetItemPosition 方法

SetItemPosition 方法用于将某个项目放置在指定的位置。其语法格式如下:

```
BOOL SetItemPosition( int nItem, POINT pt );
```

参数说明:

- nItem: 标识项目索引。
- pt: 标识项目新的位置。

(2) Arrange 方法

Arrange 方法用于设置列表项在列表中的对齐方式,其语法格式如下:

```
BOOL Arrange( UINT nCode );
```

参数说明:

- nCode: 指定项的对齐方式。
 - LVA_ALLIGNLEFT: 使项沿着窗口的左边界对齐。
 - LVA_ALLIGNTOP: 使项沿着窗口的顶端对齐。
 - LVA_DEFAULT: 使项按照列表显示的当前对齐方式(即缺省值)对齐。
 - LVA_SNAPTOGRID: 使所有图标到最近的网格位置。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 单击“Insert”/“Resource”菜单项,在打开的 Insert Resource 对话框中单击 Import 按钮,向工程中导入位图资源。
- (3) 以 CButton 类为基类派生一个新类 CListButton;以 CListCtrl 类为基类派生一个新类 CQQList。

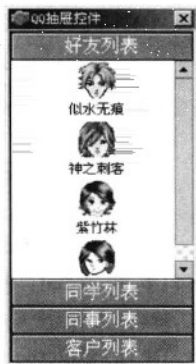


图 2.18 QQ 抽屉界面

(4) 向对话框中添加一个图片控件、一个列表视图控件和一个按钮控件。右键单击图片控件, 在弹出的菜单中选择 Properties 选项, 设置 Type 属性为 Bitmap, 设置 Image 属性为 IDB_BITMAP1。为列表视图控件关联一个 CQQList 类的对象 m_List。

(5) 在 CQQList 类中, 添加自定义函数 ShowButtonItems, 该函数用于显示指定按钮关联的列表视图项, 代码如下:

```
void CQQList::ShowButtonItems(UINT nIndex)
{
    CListButton* temp;
    temp = (CListButton*)m_pButton[nIndex];           //获得按钮指针
    m_ClientList.DeleteAllItems();                     //删除列表项
    CRect showrect = GetListClientRect();              //获得列表区域
    if(temp->m_ButtonItems.GetCount()>0)               //如果有列表项
    {
        POSITION pos, index;
        index = temp->m_ButtonIndex.GetHeadPosition(); //获得图标索引
        pos = temp->m_ButtonItems.GetHeadPosition();  //列表项索引
        CString str = temp->m_ButtonItems.GetHead();  //获得列表项文本
        CRect ClientRect;
        ClientRect = GetListClientRect();             //获得列表区域
        int m = 0, n;
        n = atoi(temp->m_ButtonIndex.GetHead());
        m_LeftMargin = showrect.Width()/2-20;         //计算左边宽度
        m_ClientList.InsertItem(m, str, n);            //插入列表项
        m_ClientList.SetItemPosition(m, CPoint(m_LeftMargin, m*(53))); //设置列表项位置
        while (pos != temp->m_ButtonItems.GetTailPosition()
            && index != temp->m_ButtonIndex.GetTailPosition())
        {
            n = atoi(temp->m_ButtonIndex.GetNext(index)); //获得下一个图标索引
            str = temp->m_ButtonItems.GetNext(pos);        //获得下一个列表项文本
            m_ClientList.InsertItem(m, str, n);            //插入列表项
            m_ClientList.SetItemPosition(m, CPoint(m_LeftMargin, m*(53))); //设置列表项位置
            m+=1;                                           //列表项索引加1
        }
        n = atoi(temp->m_ButtonIndex.GetAt(index));      //获得图标索引
        str = temp->m_ButtonItems.GetAt(pos);            //获得文本
        m_ClientList.InsertItem(m, str, n);              //插入列表项
        m_ClientList.SetItemPosition(m, CPoint(m_LeftMargin, m*(53))); //设置列表项位置
    }
}
```

举一反三

根据本实例, 读者可以:

- 实现列表视图控件的提示条。

实例 066 位图背景列表框控件

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\02\066

实例说明

在实际的应用程序中, 除了设置列表框的各种属性以外, 还可以为列表框添加位图背景。本实例实现的是位图背景的列表框控件, 运行结果如图 2.19 所示。

技术要点

本实例可以使用 GetItemRect 方法、GetTopIndex 方法和 GetText 方法来实现。

(1) GetItemRect 方法。GetItemRect 方法用于获得列表项区域, 语法如下:

```
int GetItemRect( int nIndex, LPRECT lpRect ) const;
```

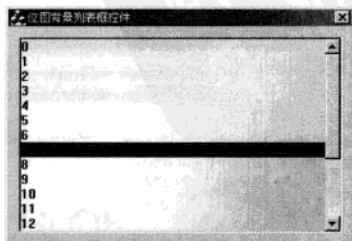


图 2.19 位图背景列表框控件

参数说明:

- nIndex: 确定项的基于零的索引。
- lpRect: 指定指向 RECT 结构的长型指针, 接收项的列表框客户区坐标。

(2) GetTopIndex 方法。GetTopIndex 方法用于获得列表框中第一个可见项的基于零的索引, 语法如下:

```
int GetTopIndex() const;
```

(3) GetText 方法。GetText 方法用于获得指定列表项的显示文本, 语法如下:

```
int GetText( int nIndex, LPTSTR lpszBuffer ) const;
void GetText( int nIndex, CString& rString ) const;
```

参数说明:

- nIndex: 指定获取的字符串的基于零的索引。
- lpszBuffer: 指向接收字符串的缓冲区的指针。缓冲区必须有足够的空间来存储字符串和终止字符。字符串大小可以通过调用 GetTextLen 成员函数提前指定。
- rString: CString 对象。

实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 以 CListBox 类为基类派生一个 CListBmp 类。
- (3) 在对话框中添加一个列表框控件, 为控件添加 CListBmp 类的成员变量 m_List, 向工程中导入一个位图资源。

(4) 在 CListBmp 类中添加 OnPaint 方法绘制列表框控件的背景位图, 代码如下:

```
void CListBmp::OnPaint()
{
    CPaintDC dc(this);
    CRect rect, clrRC;
    GetClientRect(&rect);
    CBitmap bitmap;
    CDC memdc;
    memdc.CreateCompatibleDC(&dc);
    bitmap.LoadBitmap(IDB_BITMAP1);
    memdc.SelectObject(&bitmap);
    dc.BitBlt(0,0,rect.Width(),rect.Height(),&memdc,0,0,SRCCOPY);
    bitmap.DeleteObject();
    ReleaseDC(&memdc);
    for(int i=0;i<rect.Height()/GetItemHeight(0);i++)
    {
        GetItemRect(i,clrRC);
        CString str;
        GetText(i+GetTopIndex(),str);
        dc.SetBkMode(TRANSPARENT);
        dc.TextOut(0,i*GetItemHeight(i),str);
    }
}
```

(5) 在 CListBmp 类中重写 DrawItem 虚函数, 根据列表项状态绘制列表项, 代码如下:

```
void CListBmp::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
    CDC dc;
    dc.Attach(lpDrawItemStruct->hDC);
    int nIndex = lpDrawItemStruct->itemID;
    //判断项目状态
    int nState = lpDrawItemStruct->itemState;
    CRect rect, clrRC;
    GetClientRect(&rect);
    CBitmap bitmap;
    CDC memdc;
    memdc.CreateCompatibleDC(&dc);
    bitmap.LoadBitmap(IDB_BITMAP1);
    memdc.SelectObject(&bitmap);
    GetItemRect(nIndex,clrRC);
    m_pFont = GetFont();
    dc.SelectObject(m_pFont);
```

```

if(nState & ODS_SELECTED)           //处于选中状态
{
    dc.SetTextColor(RGB(200,0,0));    //设置选中状态文本颜色
    dc.FillSolidRect(&clrRC, RGB(0,0,200)); //填充项目区域为高亮效果
}
else
{
    int nCurSel = nIndex-GetTopIndex(); //设置当前索引基于可见项的位置
    dc.BitBlt(0,nCurSel*clrRC.Height(),clrRC.Width(),clrRC.Height(),
        &memdc,0,nCurSel*clrRC.Height(),SRCCOPY); //绘制列表项背景
    dc.SetTextColor(RGB(0,0,0));      //设置文本颜色
}
if(nIndex != -1)
{
    CString str;
    GetText(nIndex,str);              //获得控件文本
    dc.SetBkMode(TRANSPARENT);        //设置背景透明
    dc.TextOut(0,(nIndex-GetTopIndex())*clrRC.Height(),str); //绘制列表项文本
}
m_pFont->DeleteObject();
bitmap.DeleteObject();
ReleaseDC(&memdc);
dc.DeleteDC();
}

```

(6) 在 CListBmp 类中添加 OnVScroll 方法, 在拖动滚动条时重绘控件, 代码如下:

```

void CListBmp::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    Invalidate();
    CListBox::OnVScroll(nSBCode, nPos, pScrollBar);
}

```

举一反三

根据本实例, 读者可以:

- 设置其他控件的字体颜色;
- 设置控件的背景颜色。

2.5 组合框控件典型实例

组合框控件是编辑框控件和列表框控件的组合控件, 它不仅可以提供用户选择的功能, 也能提供用户输入的功能。本节通过几个实例介绍组合框控件的使用。

实例 067 将数据表中的字段添加到组合框控件

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\02\067

实例说明

本实例实现将数据库中的数据添加到组合框控件中。运行程序, 程序将 Access 数据库 mrsoft.mdb 中的数据添加到组合框控件中, 数据表中单个字段的每一行对应组合框控件的一项, 实例运行结果如图 2.20 所示。

技术要点

本实例主要通过 ADO 技术连接数据库, 然后通过组合框控件的 AddString 方法添加从数据库读出的数据。AddString 方法的语法如下:

```
int AddString( LPCTSTR lpszString );
```

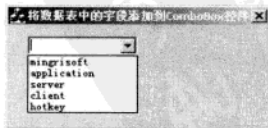


图 2.20 将数据表中的字段添加到组合框控件

实现过程

- ```

BOOL CComboBoxDlg::OnInitDialog()

```

### 举一反三

- 将 INI 文件的数据加入到组合框控件;
- 将注册表中存储的数据加入到组合框控件。

实例位置: 光盘\mingrisoft\02\068

## 实例说明

## 技术要点

图 2.21 带查询功能的 ComboBox 控件



组合框控件中查找符合要求的字符串, 如果找到符合要求的字符串就将其返回。它的语法如下:

```
int SelectString(int nStartAfter, LPCTSTR lpszString);
```

参数说明:

- nStartAfter: 指定开始查找的位置索引, 如果为-1, 从头开始查找。
- lpszString: 所要查找的字符串。

通过 CComboBox 类的 SetEditSel 方法使字符串处于选中状态, 语法如下:

```
BOOL SetEditSel(int nStartChar, int nEndChar);
```

参数说明:

- nStartChar: 开始标识的位置索引。
- nEndChar: 结束标识的位置索引。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在工程中添加新类 AutoComplete。
- (3) 在对话框上添加组合框控件, 添加成员变量 m\_sercmb, 继承自新类 AutoComplete。
- (4) 在对话框初始化时向组合框中插入数据, 代码如下:

```
BOOL CQueryComboBoxDlg::OnInitDialog()
```

```
{
 CDialog::OnInitDialog();
 //...此处代码省略
 //向组合框中插入数据
 m_sercmb.AddString("Monday");
 m_sercmb.AddString("Tuesday");
 m_sercmb.AddString("Wednesday");
 m_sercmb.AddString("Thursday");
 m_sercmb.AddString("Friday");
 m_sercmb.AddString("Saturday");
 m_sercmb.AddString("Sunday");
 return TRUE;
}
```

- (5) 在新类 AutoComplete 中添加 CBN\_EDITUPDATE 消息的实现函数, 代码如下:

```
void AutoComplete::OnEditupdate()
```

```
{
 if(!m_bAutoComplete)return;
 CString str;
 GetWindowText(str); //获得组合框的编辑框中文本
 int nLength=str.GetLength(); //获得文本长度
 DWORD dwCurSel=GetEditSel(); //获得文本的起始位置
 DWORD dStart=LOWORD(dwCurSel);
 DWORD dEnd=HIWORD(dwCurSel);
 if(SelectString(-1,str)==CB_ERR) //查找字符
 {
 SetWindowText(str); //设置显示字符串
 if(dwCurSel!=CB_ERR)
 SetEditSel(dStart,dEnd); //设置编辑框部分选中的字符串
 }
 GetWindowText(str); //获得组合框的编辑框中文本
 if(dEnd < nLength && dwCurSel!=CB_ERR)
 SetEditSel(dStart,dEnd);
 else
 SetEditSel(nLength-1);
}
```

- (6) 通过 PreTranslateMessage 处理组合框控件的 WM\_KEYDOWN 消息, 代码如下:

```
BOOL AutoComplete::PreTranslateMessage(MSG* pMsg)
```

```
{
 if(pMsg->message==WM_KEYDOWN) //如果键盘键按下
 {
 m_bAutoComplete=true;
 int nVirtKey=(int)pMsg->wParam;
 if(nVirtKey==VK_DELETE||nVirtKey==VK_BACK)//按下的是删除键或返回键
 m_bAutoComplete=false;
 }
 return CComboBox::PreTranslateMessage(pMsg);
}
```



## 举一反三

根据本实例,读者可以:

- 实现编辑框控件的自动查询功能。

## 实例 069 自动调整组合框的宽度

这是一个可以提高分析能力的实例

实例位置: 光盘\mingrisoft\02\069

## 实例说明

组合框控件的下拉列表宽度在默认情况下和组合框宽度是相同的,但是如果组合框中的字符串宽度超过了下拉列表的宽度,那么该字符串将不能完全显示。本实例通过自动调整组合框下拉列表的宽度来解决这一问题。运行程序,在组合框中添加宽度超出组合框宽度的字符串,单击组合框中的三角按钮,可以看到下拉列表已根据字符串的长度自动调整了宽度,如图 2.22 所示。

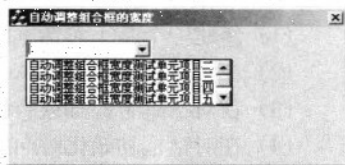


图 2.22 自动调整组合框的宽度

## 技术要点

本实例主要通过处理 OnCtlColor 消息来实现,OnCtlColor 消息主要用于处理控件的颜色,在处理该消息的函数中调用 GetSystemMetrics 函数来获得组合框控件的下拉列表宽度,然后通过 CDC 类的 GetTextExtent 方法获得字体的宽度。GetSystemMetrics 函数用于获得各种窗体尺寸,语法如下:

```
int GetSystemMetrics(int nIndex);
```

参数说明:

- nIndex: 想要获得系统配置的项目索引。主要取值如下。
  - SM\_CXCURSOR: 用户鼠标的 x 轴坐标。
  - SM\_CYCURSOR: 用户鼠标的 y 轴坐标。
  - SM\_CXSCREEN: 屏幕的宽度。
  - SM\_CYSCREEN: 屏幕的高度。

## 实现过程

- (1) 新建一个对话框应用程序。
- (2) 在工程中添加基于 CcomboBox 的新类 MyComboBox。
- (3) 在对话框上添加组合框控件,设置 ID 属性为 IDC\_RESIZECMB,添加成员变量 m\_resizecmb,继承自新类 MyComboBox。

- (4) 在新类 MyComboBox 中添加 WM\_CTLCLOR 消息的实现函数,代码如下:

```
HBRUSH MyComboBox::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
 HBRUSH hbr = CComboBox::OnCtlColor(pDC, pWnd, nCtlColor);
 switch(nCtlColor)
 {
 case CTLCLOR_EDIT:
 break;
 case CTLCLOR_LISTBOX:
 int iItemNum=GetCount(); //获得列表项数量
 int iWidth=0;
 CString strItem;
 CClientDC dc(this);
 int iSaveDC=dc.SaveDC();
 dc.SelectObject(GetFont());
 int iVSWidth=::GetSystemMetrics(SM_CXVSCROLL); //获得下拉列表宽度
 for(int i=0;i<iItemNum;i++)
 {
```

```

GetLBText(i, strItem); //获得选中的列表项文本
int iWholeWidth=dc.GetTextExtent(strItem).cx+iVSWidth;//获得显示文本宽度
iWidth=max(iWidth,iWholeWidth); //获得列表和文本中最大的宽度
}
iWidth+=dc.GetTextExtent("a").cx;
dc.RestoreDC(iSaveDC);
if(iWidth>0)
{
 CRect rc;
 pWnd->GetWindowRect(&rc); //获得窗口区域
 if(rc.Width()!=iWidth)
 {
 rc.right=rc.left+iWidth;
 pWnd->MoveWindow(&rc); //设置窗口区域
 }
}break;
}
return hbr;
}

```

(5) 在 CComboBoxResizeDlg.cpp 文件中向组合框中添加字符串数据, 代码如下:

```

BOOL CComboBoxResizeDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 //向组合框中插入数据
 m_resizecmb.AddString("自动调整组合框宽度测试单元项目一");
 m_resizecmb.AddString("自动调整组合框宽度测试单元项目二");
 m_resizecmb.AddString("自动调整组合框宽度测试单元项目三");
 m_resizecmb.AddString("自动调整组合框宽度测试单元项目四");
 m_resizecmb.AddString("自动调整组合框宽度测试单元项目五");
 return TRUE;
}

```

### 举一反三

根据本实例, 读者可以:

- 根据字符串长度调整列表视图控件中单元格的宽度。

## 实例 070 颜色组合框

这是一个可以提高分析能力的实例

实例位置: 光盘\mingrisoft\02\070

### 实例说明

默认情况下, 组合框只能显示文本信息, 本实例是对组合框功能的扩展, 实现在组合框中选择颜色。运行实例, 单击组合框中的下三角按钮弹出列表, 可以看到组合框中的每一项都由颜色矩形和颜色名称组成。实例运行结果如图 2.23 所示。

### 技术要点

为了能够在组合框中显示颜色, 需要自定义一个组合框控件, 改写 DrawItem 虚方法, 在该方法中根据项目的当前状态, 绘制相应效果的颜色和文本。为了获取当前项目的颜色值, 笔者采用的方式是利用组合框中项目的附加信息来记录颜色值, 即调用 SetItemData 方法为某个项目关联一个颜色值。然后利用 GetItemData 方法来获取该项目的颜色值。为了让用户在添加项目时能够设置项目的颜色值, 程序中提供了一个 AddItem, 用于向组合框中添加一个颜色选项。

### 实现过程

- (1) 新建一个基于对话框的应用程序。

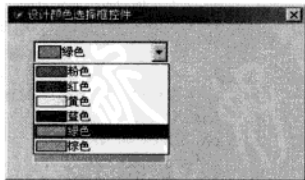


图 2.23 颜色组合框



(2) 向对话框中添加组合框和图片控件。

(3) 从 CComboBox 类派生一个子类 CColorCombox, 向该子类中添加 AddItem 方法, 用于向颜色组合框中添加颜色选项。代码如下:

```
int CColorCombox::AddItem(LPCTSTR lpszText, COLORREF clrValue)
{
 int nIndex = AddString(lpszText); //添加项目
 SetItemData(nIndex, clrValue); //设置项目的颜色值
 return nIndex; //返回新添加的项目索引
}
```

(4) 向 CColorCombox 类中添加 GetCurColor 方法, 获取当前选项的颜色值。代码如下:

```
COLORREF CColorCombox::GetCurColor()
{
 int nIndex = GetCurSel();
 if (nIndex != -1)
 {
 return GetItemData(nIndex);
 }
 else
 return -1;
}
```

(5) 改写组合框类的 DrawItem 虚方法, 根据当前项目的不同状态, 来绘制相应效果的项目。代码如下:

```
void CColorCombox::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
 ASSERT(lpDrawItemStruct->CtlType == ODT_COMBOBOX); //验证是否为组合框控件
 CDC dc;
 dc.Attach(lpDrawItemStruct->hDC);
 CRect itemRC (lpDrawItemStruct->rcItem); //获取项目区域
 CRect clrRC = itemRC; //定义显示颜色的区域
 CRect textRC = itemRC; //定义文本区域
 COLORREF clrText = GetSysColor(COLOR_WINDOWTEXT); //获取系统文本颜色
 COLORREF clrSelected = GetSysColor(COLOR_HIGHLIGHT); //选中时的文本颜色
 COLORREF clrNormal = GetSysColor(COLOR_WINDOW); //获取窗口背景颜色
 int nIndex = lpDrawItemStruct->itemID; //获取当前项目索引
 int nState = lpDrawItemStruct->itemState; //判断项目状态
 if(nState & ODS_SELECTED) //处于选中状态
 {
 dc.SetTextColor((0x00FFFFFF & ~(clrText))); //文本颜色取反
 dc.SetBkColor(clrSelected); //设置文本背景颜色
 dc.FillSolidRect(&clrRC, clrSelected); //填充项目区域为高亮效果
 }
 else
 {
 dc.SetTextColor(clrText); //设置正常的文本颜色
 dc.SetBkColor(clrNormal); //设置正常的文本背景颜色
 dc.FillSolidRect(&clrRC, clrNormal);
 }
 if(nState & ODS_FOCUS) //如果项目获取焦点, 绘制焦点区域
 {
 dc.DrawFocusRect(&itemRC);
 }
 int nclrWidth = itemRC.Width()/4; //计算文本区域
 textRC.left = nclrWidth + 1;
 clrRC.DeflateRect(2, 2); //计算颜色显示区域
 clrRC.right = nclrWidth;
 //绘制颜色文本并且填充颜色区域
 if (nIndex != -1) //项目不为空
 {
 COLORREF clrItem = GetItemData(nIndex); //获取项目颜色
 dc.SetBkMode(TRANSPARENT);
 CString szText;
 GetLBText(nIndex, szText); //获取文本
 //输出文本
 dc.DrawText(szText, textRC, DT_LEFT|DT_VCENTER|DT_SINGLELINE);
 dc.FillSolidRect(&clrRC, clrItem); //输出颜色
 dc.FrameRect(&clrRC, &CBrush(RGB(0, 0, 0))); //绘制黑色边框
 }
 dc.Detach();
}
```

(6) 在主对话框中利用类向导为组合框控件命名为 m\_ColorBox, 其类型为 CColorCombox。

(7) 在对话框初始化时向颜色组合框中添加颜色选项。代码如下:

```
m_ColorBox.AddItem("红色", RGB(255, 0, 0));
m_ColorBox.AddItem("蓝色", RGB(0, 0, 255));
m_ColorBox.AddItem("绿色", RGB(0, 255, 0));
m_ColorBox.AddItem("黄色", RGB(255, 255, 0));
m_ColorBox.AddItem("粉色", RGB(255, 0, 255));
m_ColorBox.AddItem("棕色", RGB(255, 128, 64));
```

## 举一反三

根据本实例, 读者可以:

- 实现图标组合框。

## 实例 071 多列显示的组合框

这是一个可以提高分析能力的实例

实例位置: 光盘\mingrisoft\02\071

## 实例说明

组合框在默认情况下都是单列显示, 单列显示的组合框只能提供单一的数据, 如果是多列显示就可以提供一些提示信息或一些计算结果。本实例就是实现一个多列显示的组合框, 运行程序, 单击列表框中的下三角按钮, 下拉列表变成 3 行 3 列的二维表格。程序运行结果如图 2.24 所示。

## 技术要点

本实例的实现需要覆写 CComboBox 类的 DrawItem 函数, 新建基于 CComboBox 的新类 MyComboBox, 在 MyComboBox 类中添加函数 DrawItem, DrawItem 函数实现的是单行数据的绘制, 在函数中通过 LineTo 方法来绘制直线, 在两个单元格之间绘制直线, 还需要绘制底部线条。

通过 CStringList 类来存储二维数据, 通过 CStringList 对象的下标来代表行, 通过 CStringList 类的 AddHead 方法来添加行的第一个数据, 通过 AddTail 方法向行中追加数据项。

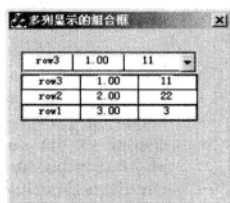


图 2.24 多列显示的组合框

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在工程中添加基于 CComboBox 的新类 MyComboBox。
- (3) 在对话框上添加组合框控件, 设置 ID 属性为 IDC\_MULCMB。
- (4) 在 MyComboBox.h 文件中添加变量声明:

```
UINT totalcol;
CStringList* ItemList;
CString item[512];
CStringList litem[5];
```

- (5) 自定义函数 init 完成组合框初始化, 代码如下:

```
void MyComboBox::init()
{
 //插入列表项
 UINT itemindex;
 itemindex=CComboBox::AddString("row1");
 litem[0].AddHead("row1");
 litem[1].AddTail("1.00");
 litem[2].AddTail("11");
 itemindex=CComboBox::AddString("row2");
 litem[0].AddHead("row2");
 litem[1].AddTail("2.00");
 litem[2].AddTail("22");
 itemindex=CComboBox::AddString("row3");
```

```
litem[0].AddHead("row3");
litem[1].AddTail("3.00");
litem[2].AddTail("3");
this->SetCurSel(0); //设置默认选中列表项
}
```

(6) 在新类 MyComboBox 中添加 DrawItem 函数, 实现组合框自己绘制的功能, 代码如下:

```
void MyComboBox::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
 CPen* pen;
 CDC* pDC=CDC::FromHandle(lpDrawItemStruct->hDC); //获得设备上下文
 CWnd* pListBox=pDC->GetWindow(); //获得窗口指针
 CRect rc1,rc2,rc3;
 CRect allrc=lpDrawItemStruct->rcItem; //获得区域
 BOOL select=lpDrawItemStruct->itemState&ODS_SELECTED; //选中状态
 BOOL focus=lpDrawItemStruct->itemAction&ODA_FOCUS; //获得焦点
 BOOL textfocus=lpDrawItemStruct->itemState&ODS_FOCUS;
 COLORREF textcolor=GetSysColor(COLOR_WINDOWTEXT);
 COLORREF selectcolor=GetSysColor(COLOR_HIGHLIGHT);
 COLORREF normalcolor=GetSysColor(COLOR_WINDOW);
 int width=allrc.Width()/3; //按提示窗口宽度分为3份
 rc1.SetRect(allrc.left,allrc.top,width,allrc.bottom); //设置第1份区域
 rc2.SetRect(rc1.right,rc1.top,2*width,allrc.bottom); //设置第2份区域
 rc3.SetRect(rc2.right,rc2.top,3*width,allrc.bottom); //设置第3份区域
 if(select)
 {
 POSITION pos=litem[0].FindIndex(lpDrawItemStruct->itemID); //查找索引
 CString str1;str1=litem[0].GetAt(pos); //获得字符
 pDC->DrawText(str1,-1,rc1,DT_CENTER); //绘制字符
 }
 else
 {
 //绘制为选中的部分
 POSITION pos=litem[0].FindIndex(lpDrawItemStruct->itemID);
 CString str1;str1=litem[0].GetAt(pos);
 pDC->DrawText(str1,-1,rc1,DT_CENTER);
 pos=litem[1].FindIndex(lpDrawItemStruct->itemID);
 CString str2;str2=litem[1].GetAt(pos);
 pDC->DrawText(str2,-1,rc2,DT_CENTER);
 pos=litem[2].FindIndex(lpDrawItemStruct->itemID);
 CString str3;str3=litem[2].GetAt(pos);
 pDC->DrawText(str3,-1,rc3,DT_CENTER);
 //绘制分隔线
 pDC->MoveTo(allrc.left,allrc.bottom-1);
 pDC->LineTo(allrc.right,allrc.bottom-1);
 pDC->MoveTo(rc1.right,rc1.top);
 pDC->LineTo(rc1.right,allrc.bottom-1);
 pDC->MoveTo(rc2.right,rc2.top);
 pDC->LineTo(rc2.right,allrc.bottom-1);
 }

 if(focus)
 {
 pDC->DrawFocusRect(allrc); //绘制焦点框
 }
}
```

(7) 在新类 MyComboBox 中覆写 AddString 函数, 调用自定义函数 AddItem 实现数据的添加, 代码如下:

```
int MyComboBox::AddString(LPCTSTR lpszString)
{
 return AddItem(lpszString,0,CComboBox::GetCount());
}
```

(8) 添加自定义函数 AddItem, 该函数用于向组合框中添加数据, 代码如下:

```
int MyComboBox::AddItem(CString strItem,int nCol,int nRow,int nMask,int nFmt)
{
 int item;
 if(nRow==CComboBox::GetCount()) //获得组合框中列表项行数
 item=CComboBox::AddString(strItem); //向组合框中插入数据
 litem[nCol].AddTail(strItem); //设置列本本
 return 1;
}
```

(9) 在对话框初始化时实现组合框数据的初始化, 代码如下:

```
BOOL CMultiComboBoxDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_mulcmb.init();
 return TRUE; }

```

### 举一反三

根据本实例, 读者可以:

- 在编辑框控件中绘制多列表。

## 实例 072 QQ 登录式的用户选择列表

这是一个可以提高分析能力的实例

实例位置: 光盘\mingrsoft\02\072

### 实例说明

用户在使用软件时, 经常可以看到带图标的组合框, 其实 Visual C++中也提供了这样的组合框。运行实例, 单击组合框中的下三角按钮弹出列表, 可以看到组合框中的每一项都由图标和名称组成。实例运行结果如图 2.25 所示。



图 2.25 带图标的组合框

### 技术要点

本实例主要通过 CcomboBoxEx 类的 InsertItem 方法来实现的, InsertItem 方法用于向扩展组合框中插入数据。语法如下:

```
int InsertItem(const COMBOBOXEXITEM* pCBItem);
```

参数说明:

- pCBItem: COMBOBOXEXITEM 结构指针, 该结构用于接收项的信息, 并包含了项的回调标志值。

返回值: 调用成功时返回插入项的下标, 否则返回-1。

### 实现过程

- 新建一个基于对话框的应用程序。
- 在对话框上添加一个扩展组合框控件, 添加 CcomboBoxEx 类成员变量 m\_Combo。
- 在头文件中声明一个图像列表对象 m\_ImageList。
- 主要程序代码如下:

```
BOOL CIconComboDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 // 系统生成的代码省略
 CString str[]={ "钱夫人", "小丹尼", "卡卡罗特", "琪琪", "特兰克斯", "贝吉塔", "天津饭" };
 m_ImageList.Create(16,16,ILC_COLOR24,ILC_MASK,1,0);
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON1)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON2)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON3)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON4)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON5)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON6)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON7)); //向图像列表中添加图标
 m_Combo.SetImageList(&m_ImageList);
 for(int i=0;i<7;i++)
 {
 COMBOBOXEXITEM cbi;
 cbi.mask = CBEIF_IMAGE|CBEIF_INDENT|CBEIF_OVERLAY|

```



```

 CBEIF_SELECTEDIMAGE|CBEIF_TEXT;
 cbi.iItem = i;
 cbi.pszText = str[i].GetBuffer(0); //设置列表项文本
 cbi.cchTextMax = str[i].GetLength(); //设置文本最大长度
 cbi.ilImage = i; //设置图标索引
 cbi.iSelectedImage = i; //设置选中图标索引
 cbi.iOverlay = 0;
 cbi.ilIndent = (0 & 0x03);
 m_Combo.InsertItem(&cbi); //插入数据
 }
 return TRUE;
}

```

### 举一反三

根据本实例，读者可以：

- 实现图标组合框。

## 实例 073 显示系统盘符组合框

这是一个可以提高分析能力的实例

实例位置：光盘\mingrisoft\02\073

### 实例说明

组合框在默认情况下都是用户自己添加数据，但是有些时候需要在不同的计算机中运行程序，那么，系统的盘符信息就是不固定的，为了解决这个问题，可以使用组合框的方法查找系统盘符，并显示出来。本实例就是实现显示系统盘符的组合框。程序运行结果如图 2.26 所示。

### 技术要点

本实例中使用了 GetLogicalDriveStrings 函数，该函数的作用是将系统中合法的盘符添加到字符缓冲区中。语法如下：

```
DWORD GetLogicalDriveStrings(DWORD nBufferLength, LPTSTR lpBuffer);
```

参数说明：

- nBufferLength：表示字符缓冲区的长度。
- lpBuffer：表示字符缓冲区。

返回值：如果函数执行成功，返回值是复制到缓冲区中的字节数，不包括终止符。如果函数执行失败，返回值为 0。

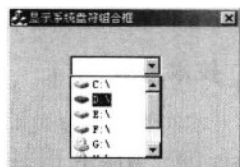


图 2.26 显示系统盘符组合框

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在工程中添加一个扩展组合框控件，关联成员变量 m\_ComboEx。
- (3) 主要程序代码如下：

```

void CComboCatalogDlg::LoadSysDisk()
{
 m_ComboEx.SetImageList(&m_ImageList);
 m_ComboEx.ResetContent();
 char pchDrives[128] = {0};
 char* pchDrive;
 GetLogicalDriveStrings(sizeof(pchDrives), pchDrives); //列举盘符
 pchDrive = pchDrives;
 int nItem = 0;
 while(*pchDrive)
 {
 COMBOBOXEXITEM cbi;
 CString csText;
 cbi.mask = CBEIF_IMAGE|CBEIF_INDENT|CBEIF_OVERLAY|
 CBEIF_SELECTEDIMAGE|CBEIF_TEXT;
 }
}

```

```

SHFILEINFO shInfo; //定义文件信息
int nlcon;
SHGetFileInfo(pchDrive, 0, &shInfo, sizeof(shInfo),
 SHGFI_ICON|SHGFI_SMALLICON); //获取系统文件图标
nlcon = shInfo.iIcon;
//设置COMBOBOXEXITEM结构
cbi.iItem = nlcon;
cbi.pszText = pchDrive;
cbi.cchTextMax = strlen(pchDrive);
cbi.iImage = nlcon;
cbi.iSelectedImage = nlcon;
cbi.iOverlay = 0;
cbi.iIndent = (0 & 0x03);
m_ComboEx.InsertItem(&cbi); //插入数据
nlcon++;
pchDrive += strlen(pchDrive) + 1;
}
}

```

### 举一反三

根据本实例，读者可以：

- 在组合框中显示查找的文件。

## 2.6 列表视图控件典型实例

列表视图控件可以用来浏览数据，也可以用来显示图标，在开发应用程序时经常要使用列表视图控件，可以用它来代替 DataGrid 数据控件来浏览数据，本节通过几个实例介绍列表视图控件的应用。

### 实例 074 Windows 资源管理器

本实例可以方便操作、提高效率

实例位置：光盘\mingrisoft\02\074

#### 实例说明

在 Windows 的资源管理器中，使用了列表视图控件来显示文件和目录信息。在该列表视图控件中显示了当前目录下的所有子目录和文件，用户可以根据文件名、文件日期等信息进行排序，并且双击某个文件时会调用系统关联的程序打开文件。本实例通过列表视图控件设计 Windows 资源管理器，实例运行结果如图 2.27 所示。

#### 技术要点

为了能够在列表视图中显示文件和目录，在列表视图控件中定义了一个字符串变量 m\_CurDir，表示列表视图控件当前显示的目录文件。然后定义一个 DisplayPath 函数，根据参数描述的目录，列举该目录下的文件和目录，将其显示在列表视图中。



图 2.27 Windows 资源管理器

#### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 从 CHeaderCtrl 派生一个子类 CFileHeaderCtrl，从 CListCtrl 类派生一个子类 CFileListCtrl。
- (3) 在对话框上添加一个图片控件和一个列表视图控件，设置图片控件 ID 属性为 IDC\_

FRAME; 设置列表视图控件 View 属性为 Report; 添加列表视图控件, 分别为控件添加成员变量 m\_Frame 和 m\_FileList。

(4) 向 CFileListCtrl 类中添加成员变量。代码如下:

```
CString m_BaseDir; //基目录
CString m_CurDir; //记录当前列表中文件的目录
CFileHeaderCtrl m_ctlHeader; //列头
int m_nNumColumns; //列数
int m_nSortColumn; //排序列
BOOL m_bAscend; //是否升序排列
CImageList m_ImageList; //定义图像列表
```

(5) 向 CFileListCtrl 类中添加 SetColumns 方法, 为视图列表添加列。代码如下:

```
BOOL CFileListCtrl::SetColumns(const CString &strHeadings)
{
 int nStart = 0;
 for(;;)
 {
 int nComma = strHeadings.Find(_T(','), nStart); //截取字符串
 if(nComma == -1)
 break;
 CString strHeading = strHeadings.Mid(nStart, nComma - nStart); //获取文本
 nStart = nComma + 1; //略过","
 int nSemiColon = strHeadings.Find(_T(';'), nStart);
 if(nSemiColon == -1)
 nSemiColon = strHeadings.GetLength();
 int nWidth = atoi(strHeadings.Mid(nStart, nSemiColon - nStart)); //获取宽度
 nStart = nSemiColon + 1; //指向下一列信息
 if(InsertColumn(m_nNumColumns++, strHeading, LVCFMT_LEFT, nWidth) == -1)
 return FALSE; //插入列
 }
 return TRUE;
}
```

(6) 向 CFileListCtrl 中添加 AddItem 方法, 向视图列表中添加视图项, 并且为视图项关联数据 (当前行所有列的文本)。代码如下:

```
int CFileListCtrl::AddItem(LPCTSTR pszText, ...)
{
 int nIndex = InsertItem(GetItemCount(), pszText); //添加行, 返回行索引
 LPTSTR* pszColumnTexts = new LPTSTR[m_nNumColumns]; //记录各列文本
 pszColumnTexts[0] = new TCHAR[strlen(pszText) + 1];
 strcpy(pszColumnTexts[0], pszText); //复制第一列文本到pszColumnTexts中
 va_list list;
 va_start(list, pszText);
 //设置列文本
 for(int nColumn = 1; nColumn < m_nNumColumns; nColumn++) //将各列文本复制到pszColumnTexts中
 {
 pszText = va_arg(list, LPCTSTR);
 CListCtrl::SetItem(nIndex, nColumn, LVIF_TEXT, pszText, 0, 0, 0, 0);
 pszColumnTexts[nColumn] = new TCHAR[strlen(pszText) + 1];
 strcpy(pszColumnTexts[nColumn], pszText);
 }
 va_end(list);
 SetItemDataList(nIndex, pszColumnTexts); //设置视图项数据
 return nIndex;
}
```

(7) 向 CFileListCtrl 中添加 SetItemDataList 方法, 设置视图项关联的数据。代码如下:

```
BOOL CFileListCtrl::SetItemDataList(int item, LPTSTR *pchTexts)
{
 if(CListCtrl::GetItemData(item) == NULL)
 {
 CItemData* pItemData = new CItemData(); //创建一个CItemData对象
 pItemData->m_ColumnTexts = pchTexts; //设置列文本
 return CListCtrl::SetItemData(item, (DWORD)pItemData); //设置项目数据
 }
}
```

(8) 向 CFileListCtrl 中添加 LoadSysFileIcon 方法, 加载系统文件图像列表。代码如下:

```
void CFileListCtrl::LoadSysFileIcon()
{
 SHFILEINFO shInfo; //定义外壳文件信息
 memset(&shInfo, 0, sizeof(SHFILEINFO));
```

```
HIMAGELIST hImage = (HIMAGELIST)SHGetFileInfo("C:\\",0,&shInfo, sizeof(SHFILEINFO),
 SHGFI_SYSICONINDEX | SHGFI_SMALLICON); //加载系统文件图像列表
m_ImageList.Attach(hImage); //附加图像列表句柄
SetImageList(&m_ImageList, LVSIL_SMALL); //设置列表视图关联的图像列表控件
```

(9) 向 CFileListCtrl 类中添加 DisplayPath 方法, 显示指定目录下的子目录和文件到列表中。

代码如下:

```
void CFileListCtrl::DisplayPath(LPCTSTR lpPath)
{
 DeleteAllItems(); //删除列表视图中的所有视图项
 BOOL bFind;
 CFileFind flFind; //定义文件查找对象
 CString csPath = lpPath;
 m_CurDir = lpPath;
 if (csPath.Right(1) != "\\")
 {
 csPath += "\\";
 m_CurDir += "\\";
 }
 csPath += "*.*";
 bFind = flFind.FindFile(csPath); //开始查找文件
 CString csText, csFileSize, csDataTime;
 //设置列表当前显示的目录
 while (bFind)
 {
 bFind = flFind.FindNextFile(); //查找下一个文件
 if (!flFind.IsDots() && !flFind.IsHidden())//判断文件的属性
 {
 _int64 lFileLen = flFind.GetLength64();//获取文件长度

 if (flFind.IsDirectory()) //是否是目录
 {
 csFileSize = "文件夹";
 }
 else
 {
 //格式化文件大小
 double fGB = lFileLen / (double)(1024*1024*1024);
 if (fGB < 1)
 {
 double fMB = lFileLen / (double)(1024*1024);
 if (fMB < 1)
 {
 double fKB = lFileLen / (double)(1024);
 if (fKB > 1)
 {
 csFileSize.Format("%.2f KB", fKB);
 }
 else
 {
 csFileSize.Format("%i B", lFileLen);
 }
 }
 else
 {
 csFileSize.Format("%.2f MB", fMB);
 }
 }
 else
 {
 csFileSize.Format("%.2f GB", fGB);
 }
 }
 }

 csText = flFind.GetFileName(); //获取文件名
 CTime time;
 flFind.GetCreationTime(time); //获取文件创建时间
 csDataTime = time.Format("%Y-%m-%d %H:%M");
 //将文件名, 文件大小, 文件创建时间添加到列表视图中
 int nItem = AddItem(csText, csFileSize, csDataTime);
 SHFILEINFO shInfo; //设置文件显示的图标
 int nIcon = 0;
 SHGetFileInfo(flFind.GetFilePath(), 0, &shInfo, sizeof(shInfo), SHGFI_ICON | SHGFI_SMALLICON);
 DestroyIcon(shInfo.hIcon);
 nIcon = shInfo.iIcon;
 SetItem(nItem, 0, LVIF_IMAGE, "", nIcon, 0, 0, 0); //显示文件关联的图标
 if (flFind.IsDirectory()) //设置项目标记, 0表示文件, 1表示目录
 }
}
```



```

{
 SetItemData(nItem, 1); //设置视图项关联的数据
}
else
{
 SetItemData(nItem, 0);
}
nItem++;
}
}

```

### 举一反三

根据本实例，读者可以：

- 将数据表中数据添加到列表视图控件。

## 实例 075

### 利用列表视图控件浏览数据

本实例可以方便操作、提高效率

实例位置：光盘\mingrsoft\02\075

### 实例说明

在开发应用程序时经常使用列表控件来显示二维数据。本实例实现了用列表控件显示数据表中数据的功能。运行程序，在列表控件中显示 Access 数据库 mrdb.mdb 的 info 表中的数据。实例运行结果如图 2.28 所示。

### 技术要点

本实例主要通过 ADO 技术来连接数据库，通过 CListCtrl 类的 SetItemText 方法将数据显示在列表控件中。在使用 SetItemText 方法前，要先通过 InsertColumn 方法添加列，通过 InsertItem 方法来添加行，否则直接调用 SetItemText 方法将出错，SetItemText 方法的语法如下：

```
BOOL SetItemText(int nItem, int nSubItem, LPCTSTR lpszText);
```

参数说明：

- nItem：行索引。
- nSubItem：列索引。
- lpszText：字符串指针。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框上添加列表视图控件，设置 ID 属性为 IDC\_DATA LIST，添加成员变量 m\_datalist。
- (3) 在 StdAfx.h 文件中导入 ADO 动态链接库。
- (4) 主要程序代码如下：

```

BOOL CAddDateToListDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 //...此处代码省略
 //设置列表视图控件的风格
 m_datalist.ModifyStyle(0L, LVS_REPORT);
 m_datalist.ModifyStyle(0L, LVS_SINGLESEL);
 m_datalist.ModifyStyle(0L, LVS_SHOWSELALWAYS);
 m_datalist.ModifyStyle(0L, LVS_NOSORTHEADER);
}

```

| 姓名  | 性别 | 出生日期     | 工作单位 | 固定电话        | 移动电话        |
|-----|----|----------|------|-------------|-------------|
| 李+伟 | 男  | 19780102 | 明日科技 | 13134435*** | 04347040*** |
| 张+世 | 男  | 19721130 | 宏+集团 | 13556895*** | 04315607*** |
| 王+迎 | 女  | 19850728 | 东+电力 | 13066542*** | 04317956*** |

图 2.28 利用列表视图控件浏览数据

```

m_dataList.SetExtendedStyle(LVS_EX_GRIDLINES);
//设置列标题
m_dataList.InsertColumn(0,"姓名");
m_dataList.InsertColumn(1,"性别");
m_dataList.InsertColumn(2,"出生日期");
m_dataList.InsertColumn(3,"工作单位");
m_dataList.InsertColumn(4,"移动电话");
m_dataList.InsertColumn(4,"固定电话");
//设置列宽度
m_dataList.SetColumnWidth(0,100);
m_dataList.SetColumnWidth(1,50);
m_dataList.SetColumnWidth(2,100);
m_dataList.SetColumnWidth(3,100);
m_dataList.SetColumnWidth(4,100);
m_dataList.SetColumnWidth(5,100);
::CoInitialize(NULL); //初始化COM环境
m_pConnection=NULL; //连接对象实例化
m_pConnection.CreateInstance(__uuidof(Connection));
m_pConnection->ConnectionString="uid=;pwd=;DRIVER=
{Microsoft Access Driver (*.mdb)};DBQ=mrdb.mdb;"; //设置连接字符串
m_pConnection->Open(L"",L"",L"",adCmdUnspecified); //打开数据库
_bstr_t bstrSQL="select * from info"; //设置查询语句
m_pRecordset=m_pConnection->Execute(bstrSQL,NULL,adCmdText); //执行查询语句
int i=0;
while(!m_pRecordset->adoEOF)
{
//获得数据库中数据
xm=(char*)(_bstr_t)m_pRecordset->GetCollect("xm");
xb=(char*)(_bstr_t)m_pRecordset->GetCollect("xb");
csrq=(char*)(_bstr_t)m_pRecordset->GetCollect("csrq");
gzdw=(char*)(_bstr_t)m_pRecordset->GetCollect("gzdw");
yddh=(char*)(_bstr_t)m_pRecordset->GetCollect("yddh");
gddh=(char*)(_bstr_t)m_pRecordset->GetCollect("gddh");
m_dataList.InsertItem(i,""); //向列表视图控件中插入行
//为每一行插入列
m_dataList.SetItemText(i,0,xm);
m_dataList.SetItemText(i,1,xb);
m_dataList.SetItemText(i,2,csrq);
m_dataList.SetItemText(i,3,gzdw);
m_dataList.SetItemText(i,4,yddh);
m_dataList.SetItemText(i,5,gddh);
i++;
m_pRecordset->MoveNext(); //向下移动记录集指针
}
m_pRecordset->Close();
m_pConnection->Close();
m_pRecordset=NULL;
m_pConnection=NULL;
::CoUninitialize();
return TRUE;
}

```

### 举一反三

根据本实例，读者可以：

- 利用 INI 文件向列表控件中添加数据；
- 利用有规律的文本文件向列表控件中添加数据。

### 实例 076

## 利用列表视图控件制作 导航界面

本实例可以美化界面、简化操作

实例位置：光盘\mingrisoft\02\076

### 实例说明

列表控件能够响应鼠标的双击事件，在设计程序时，如果利用列表控件制作导航界面，会

使程序更具有特色。运行程序,双击列表中的项,列表项所对应的复选框就会被选中,实例运行结果如图 2.29 所示。

### 技术要点

本实例主要通过处理列表控件的 NM\_DBLCLK 消息来完成,如果用户双击列表控件后,NM\_DBLCLK 消息就会产生,通过 GetSelectionMark 方法可以获得列表所选项的索引,通过返回的索引值的不同可以进行相应的处理,如果返回的索引值为-1,表示没有选中列表项。

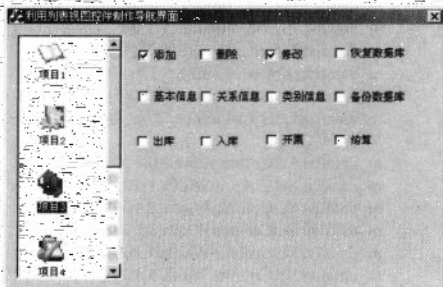


图 2.29 利用列表视图控件制作导航界面

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框上添加列表视图控件,设置 ID 属性为 IDC\_IMAGELST,添加成员变量 m\_mylist,添加 12 个复选框控件。
- (3) 向工程中添加 7 个图标资源,并修改图标 ID。
- (4) 在对话框初始化时实现对列表框中数据的初始化,代码如下:

```
BOOL CListNaviDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 //...此处代码省略
 m_imglst.Create(32,32,ILC_COLOR32|ILC_MASK,0,0); //创建图像列表
 //加载图标资源
 m_imglst.Add(::AfxGetApp()->LoadIcon(IDI_CK));
 m_imglst.Add(::AfxGetApp()->LoadIcon(IDI_CP));
 m_imglst.Add(::AfxGetApp()->LoadIcon(IDI_DB));
 m_imglst.Add(::AfxGetApp()->LoadIcon(IDI_DY));
 m_imglst.Add(::AfxGetApp()->LoadIcon(IDI_KC));
 m_imglst.Add(::AfxGetApp()->LoadIcon(IDI_PD));
 m_imglst.Add(::AfxGetApp()->LoadIcon(IDI_RK));
 m_mylist.SetImageList(&m_imglst,TVSIL_NORMAL); //关联图像列表
 //向列表视图控件中插入列表项
 m_mylist.InsertItem(0,"项目1",0);
 m_mylist.InsertItem(1,"项目2",1);
 m_mylist.InsertItem(2,"项目3",2);
 m_mylist.InsertItem(3,"项目4",3);
 m_mylist.InsertItem(4,"项目5",4);
 m_mylist.InsertItem(5,"项目6",5);
 m_mylist.InsertItem(6,"项目7",6);
 return TRUE;
}
```

- (5) 在 OnDblclkImagelst 中完成通过双击列表项实现某些功能,代码如下:

```
void CListNaviDlg::OnDblclkImagelst(NMHDR* pNMHDR, LRESULT* pResult)
{
 int cursel=m_mylist.GetSelectionMark(); //获得选中的列表项索引
 if(cursel==-1)return;
 //根据双击列表中不同的项目选中复选框
 switch(cursel)
 {
 case 0:
 {
 CButton *p=(CButton*)GetDlgItem(IDC_ADD);
 p->SetCheck(1);
 break;
 }
 case 1:
 {
 CButton *p=(CButton*)GetDlgItem(IDC_RELATINFO);
 p->SetCheck(1);
 break;
 }
 //...此处代码省略
 }
}
```



```
*pResult = 0;
```

### 举一反三

根据本实例，读者可以：

- 利用列表视图开发导航界面。

## 实例 077 在列表视图中拖动视图项

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\02\077

### 实例说明

Windows 系统是支持拖曳操作的系统，拖曳操作可以简化用户的操作步骤，一个拖曳操作可以实现剪切和复制两个操作。本实例实现了在列表视图控件中拖曳数据项的功能。运行程序，可以将列表中的任意行数据进行拖曳操作，实例运行结果如图 2.30 所示。

### 技术要点

本实例的实现主要是处理列表视图的 LVN\_BEGINDRAG 消息。LVN\_BEGINDRAG 消息在有拖曳动作发生时产生。在实现列表项的拖动功能时，首先需要调用 CreateDragImage 方法创建一个拖动的图像列表 (CImageList)，然后调用图像列表的 DragEnter 方法锁定窗口更新，在拖动过程中显示图像。接着在鼠标移动的过程中调用图像列表的 DragMove 方法移动图像，显示拖动的效果。



图 2.30 在列表视图中拖动视图项

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 以 CListCtrl 类为基类派生一个 CDragList 类。
- (3) 向对话框中添加一个列表视图控件，为控件添加一个 CDragList 类的成员变量 m\_Grid。
- (4) 在 CDragList 类的头文件中声明变量，代码如下：

```
int m_ItmIndex; //拖动项索引
CImageList* m_pDrgImg; //拖动图像列表
BOOL m_Drag; //是否拖动
```

- (5) 在 CDragList 类中添加 OnBeginDrag 方法，在开始拖动时调用，代码如下：

```
void CDragList::OnBeginDrag(NMHDR* pNMHDR, LRESULT* pResult)
{
 HD_NOTIFY * phdn = (HD_NOTIFY *) pNMHDR;
 POINT pt;
 m_ItmIndex = ((NM_LISTVIEW *) pNMHDR)->iItem; //获得当前拖动项索引
 m_pDrgImg = CreateDragImage(m_ItmIndex, &pt); //创建拖动图像列表
 m_pDrgImg->BeginDrag(0, pt); //开始拖动
 ClientToScreen(&pt); //转换客户坐标到屏幕坐标
 m_pDrgImg->DragEnter(this, pt); //锁定窗口更新
 m_Drag = TRUE; //标记拖动
 *pResult = 0;
}
```

- (6) 在 CDragList 类的消息映射部分添加映射宏，在用户拖动节点时执行 OnBeginDrag 方法，代码如下：

```
ON_NOTIFY_REFLECT(LVN_BEGINDRAG, OnBeginDrag)
```

- (7) 处理 CDragList 类的 WM\_MOUSEMOVE 消息，在该消息的处理函数中随着鼠标的移



动位置更新拖动图像的位置,代码如下:

```
void CDragList::OnMouseMove(UINT nFlags, CPoint point)
{
 if(m_Drag) //如果拖动
 {
 CPoint pt;
 pt.x = point.x;
 pt.y = point.y + (m_ItmIndex + 1) * 15; //设置移动位置
 m_pDrgImg->DragMove(pt); //移动图像
 }
 CListCtrl::OnMouseMove(nFlags, point);
}
```

(8) 处理 CDragList 类的 WM\_LBUTTONDOWN 消息,在该消息的处理函数中结束拖动,并将拖动数据添加到视图中,代码如下:

```
void CDragList::OnLButtonDown(UINT nFlags, CPoint point)
{
 if(m_Drag) //如果拖动
 {
 m_pDrgImg->EndDrag(); //结束拖动
 m_Drag = FALSE; //标记为不拖动
 char name[256];
 LV_ITEM lvi;
 CString subitem[3];
 for(int i=2;i>=0;i--)
 {
 ZeroMemory(&lvi,sizeof(LV_ITEM)); //整理内存控件
 lvi.iItem = m_ItmIndex; //设置行
 lvi.iSubItem = i; //设置列
 lvi.mask = LVIF_IMAGE | LVIF_TEXT; //标志
 lvi.pszText = name; //列表项内容
 lvi.cchTextMax = 255; //文本最大值
 GetItem(&lvi); //获得节点
 subitem[i].Format("%s",name); //设置显示文本
 }
 //插入列表项
 InsertItem(&lvi);
 SetItemText(m_ItmIndex,1,subitem[1]);
 SetItemText(m_ItmIndex,2,subitem[2]);
 }
 CListCtrl::OnLButtonDown(nFlags, point);
}
```

### 举一反三

根据本实例,读者可以:

- 在文本框之间实现文本的拖动。

## 实例 078

### 具有排序功能的列表视图控件

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\078

### 实例说明

列表控件在默认情况下不会对单击列标题产生任何动作。本实例实现了利用列标题对列表视图进行数据排序的功能。运行程序,单击列表控件列标题后,程序将对该列标题所在列的数据进行排序,如图 2.31 所示。

### 技术要点

对列表视图进行排序是通过两部分进行的。第一部分是定义一个表头控件,实现排序箭头的绘制,第二部分是通过自定义的 SortFunction 方法实现对视图列排序。

为了能够在视图列排序时在表头部分显示一个箭头标记表示当前的排列方式,需要自定义一个表头控件,根据升序或降序排列绘制不同方向的箭头标

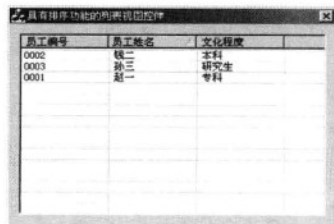


图 2.31 具有排序功能的列表视图控件

记。在列表视图控件中，表头部分是一个 CHeaderCtrl 控件，为了自定义表头，可以从 CHeaderCtrl 派生一个子类，然后改写 DrawItem 虚方法，根据不同的排列方式绘制相应的箭头符号。最后改写列表视图控件的 PreSubclassWindow 虚方法，在该方法中将自定义的表头控件子类化，使其关联到列表视图控件的表头控件上，这样就实现了列表视图控件表头部分的绘制。

```
void CListHeaderCtrl::PreSubclassWindow()
{
 ASSERT(GetStyle() & LVS_REPORT); //判断列表视图的风格是否是表格形式
 CListCtrl::PreSubclassWindow();
 VERIFY(m_ctlHeader.SubclassWindow(GetHeaderCtrl()->GetSafeHwnd()));//实现表头控件的子类化
}
```

有关表头控件的设计请参考实现过程部分。接下来介绍列表视图排序功能的实现。列表视图控件提供了 SortItems 方法用于使用自定义的方式进行排序，该方法语法如下：

```
BOOL SortItems(PFNLVCOMPARE pfnCompare, DWORD_PTR dwData);
```

参数说明：

● pfnCompare：表示用户定义的比较函数。函数原型如下：

```
int CALLBACK CompareFunc(LPPARAM lParam1, LPARAM lParam2, LPARAM lParamSort);
```

其中，lParam1 和 lParam2 表示待比较的两个选项，它们分别关联于两个视图项的数据值（调用 SetItemData 方法为视图项设置的数据值）。lParamSort 表示由 SortItems 函数传递的 dwData 参数值。比较函数 CompareFunc 的返回值是非常关键的，如果为负数，表示第一项位于第二项的前方，为正数，表示第一项位于第二项之后，为 0，表示两项相等。

● dwData：表示传递到比较函数 pfnCompare 的参数值（lParamSort 参数）。

在进行自定义的比较时，需要在比较函数中定义比较规则。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在工程中添加基于 CListCtrl 的新类 MyListCtrl。
- (3) 以 CHeaderCtrl 类为基类派生一个 CListHeader 类，以 CListCtrl 类为基类派生一个 CListHeaderCtrl 类。

(4) 向对话框中添加一个列表视图控件，为控件添加 CListHeaderCtrl 类的成员变量 m\_List。

(5) 向 CListHeader 类中添加成员变量。代码如下：

```
int m_nSortColumn; //排序列
BOOL m_bAscend; //是否为升序
```

(6) 向 CListHeader 类中添加 SetSortColumn 方法，用于设置排序列和排序方式。在绘制表头时将根据这些信息来确定在哪个列上绘制什么样式的箭头。代码如下：

```
void CFileHeaderCtrl::SetSortColumn(int nColumn, BOOL bAscend)
{
 m_nSortColumn = nColumn; //记录排序列
 m_bAscend = bAscend; //记录排序方式，升序(TRUE)还是降序(FALSE)
 HD_ITEM hItem;
 hItem.mask = HDI_FORMAT;
 GetItem(nColumn, &hItem); //获取列信息
 hItem.fmt |= HDF_OWNERDRAW; //设置自绘风格
 SetItem(nColumn, &hItem); //设置列信息
 Invalidate(); //更新表头
}
```

(7) 在 CFileHeaderCtrl 类中改写 DrawItem 虚方法，根据排序列和排序方式绘制表头。代码如下：

```
void CListHeader::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
 CDC dc; //定义设备上下文
 dc.Attach(lpDrawItemStruct->hDC); //附加设备上下文句柄
 const int nSavedIndex = dc.SaveDC(); //保存设备上下文
 CRect rc(lpDrawItemStruct->rcItem); //获取当前列区域
 CBrush brush(GetSysColor(COLOR_3DFACE)); //定义背景画刷
 dc.FillRect(rc, &brush); //填充画刷
 TCHAR szText[256];
 HD_ITEM hItem;
 hItem.mask = HDI_TEXT | HDI_FORMAT;
 hItem.pszText = szText;
```

```

hditem.cchTextMax = 255;
GetItem(lpDrawItemStruct->itemID, &hditem); //获取当前的项目信息
//设置绘制的文本格式
UINT uFormat = DT_SINGLELINE | DT_NOPREFIX | DT_NOCLIP | DT_VCENTER | DT_END_ELLIPSIS;
if(hditem.fmt & HDF_CENTER)
 uFormat |= DT_CENTER;
else if(hditem.fmt & HDF_RIGHT)
 uFormat |= DT_RIGHT;
else
 uFormat |= DT_LEFT;
if(lpDrawItemStruct->itemState == ODS_SELECTED)
{
 rc.left++;
 rc.top += 2;
 rc.right++;
}
CRect rclcon(lpDrawItemStruct->rcItem);
const int iOffset = (rclcon.bottom - rclcon.top) / 4;
if(lpDrawItemStruct->itemID == (UINT) m_nSortColumn)
 rc.right = 3 * iOffset;
rc.left += iOffset;
rc.right -= iOffset;
//绘制列文本
if(rc.left < rc.right)
 dc.DrawText(szText, -1, rc, uFormat);
//绘制箭头
if(lpDrawItemStruct->itemID == (UINT) m_nSortColumn)
{
 //定义画笔
 CPen penLight(PS_SOLID, 1, GetSysColor(COLOR_3DHILIGHT));
 CPen penShadow(PS_SOLID, 1, GetSysColor(COLOR_3DSHADOW));
 CPen* pOldPen = dc.SelectObject(&penLight); //选中画笔
 if(m_bAscend)
 {
 //绘制向上的箭头
 dc.MoveTo(rclcon.right - 2 * iOffset, iOffset);
 dc.LineTo(rclcon.right - iOffset, rclcon.bottom - iOffset - 1);
 dc.LineTo(rclcon.right - 3 * iOffset - 2, rclcon.bottom - iOffset - 1);
 dc.SelectObject(&penShadow);
 dc.MoveTo(rclcon.right - 3 * iOffset - 1, rclcon.bottom - iOffset - 1);
 dc.LineTo(rclcon.right - 2 * iOffset, iOffset - 1);
 }
 else
 {
 //绘制向下的箭头
 dc.MoveTo(rclcon.right - iOffset - 1, iOffset);
 dc.LineTo(rclcon.right - 2 * iOffset - 1, rclcon.bottom - iOffset);
 dc.SelectObject(&penShadow);
 dc.MoveTo(rclcon.right - 2 * iOffset - 2, rclcon.bottom - iOffset);
 dc.LineTo(rclcon.right - 3 * iOffset - 1, iOffset);
 dc.LineTo(rclcon.right - iOffset - 1, iOffset);
 }
 dc.SelectObject(pOldPen); //恢复原来选择的画笔
}
dc.RestoreDC(nSavedIndex); //恢复之前的设备上下文
dc.Detach(); //从设备上下文中分离设备上下文句柄
}

```

(8) 在 CListHeaderCtrl 类的头文件中声明变量, 代码如下:

```

CListHeader m_ctlHeader; //列头
int m_nNumColumns; //列数
int m_nSortColumn; //排序列
BOOL m_bAscend; //是否升序排列

```

(9) 添加 SetColumnNum 方法用于设置列表列数, 具体实现代码如下:

```

void CListHeaderCtrl::SetColumnNum(int num)
{
 m_nNumColumns = num;
}

```

(10) 向 CListHeaderCtrl 类中添加 AddItem 方法, 向视图列表中添加视图项, 并且为视图项关联数据 (当前行所有列的文本)。代码如下:

```

int CListHeaderCtrl::AddItem(LPCTSTR pszText, ...)
{
 int nIndex = InsertItem(GetItemCount(), pszText); //添加行, 返回行索引
 LPTSTR* pszColumnTexts = new LPTSTR[m_nNumColumns]; //记录各列文本
 pszColumnTexts[0] = new TCHAR[strlen(pszText) + 1];
 lstrcpyn(pszColumnTexts[0], pszText); //复制第一列文本到pszColumnTexts中
 va_list list;
}

```

```

va_start(list, pszText);
//设置列文本
for(int nColumn = 1; nColumn < m_nNumColumns; nColumn++) //将各列文本复制到pszColumnTexts中
{
 pszText = va_arg(list, LPCTSTR);
 CListCtrl::SetItem(nIndex, nColumn, LVIF_TEXT, pszText, 0, 0, 0, 0);
 pszColumnTexts[nColumn] = new TCHAR[strlen(pszText) + 1];
 lstrcpy(pszColumnTexts[nColumn], pszText);
}
va_end(list);
SetItemDataList(nIndex, pszColumnTexts); //设置视图项数据
return nIndex;
}

```

(11) 向 CListHeaderCtrl 类中添加 SetItemDataList 方法, 用于设置视图项关联的数据。代码如下:

```

BOOL CListHeaderCtrl::SetItemDataList(int iItem, LPTSTR *pchTexts)
{
 if (CListCtrl::GetItemData(iItem) == NULL)
 {
 CItemData* pItemData = new CItemData(); //创建一个CItemData对象
 pItemData->m_ColumnTexts = pchTexts; //设置列文本
 return CListCtrl::SetItemData(iItem, (DWORD)pItemData); //设置项目数据
 }
}

```

(12) 处理 LVN\_COLUMNCLICK 消息, 在该消息的处理函数中调用函数设置排序的列, 代码如下:

```

void CListHeaderCtrl::OnColumnClick(NMHDR* pNMHDR, LRESULT* pResult)
{
 NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
 int nColumn = pNMListView->iSubItem;
 Sort(nColumn, nColumn == m_nSortColumn ? !m_bAscend : TRUE);
 *pResult = 0;
}

```

(13) 添加 Sort 函数, 用于设置排序的列, 代码如下:

```

void CListHeaderCtrl::Sort(int iColumn, BOOL bAscending)
{
 m_nSortColumn = iColumn;
 m_bAscend = bAscending;
 m_ctlHeader.SetSortColumn(m_nSortColumn, m_bAscend); //设置排序的列
 SortItems(SortFunction, (DWORD)this); //调用比较函数
}

```

(14) 定义一个用于比较的函数, 其实现代码如下:

```

int CALLBACK CListHeaderCtrl::SortFunction(LPARAM IParam1, LPARAM IParam2, LPARAM IParamData)
{
 CListHeaderCtrl* pListCtrl = (CListHeaderCtrl*)(IParamData);
 CItemData* pParam1 = (CItemData*)(IParam1); //获取视图项关联的数据
 CItemData* pParam2 = (CItemData*)(IParam2);
 LPCTSTR pszText1 = pParam1->m_ColumnTexts[pListCtrl->m_nSortColumn]; //获取排序列的文本
 LPCTSTR pszText2 = pParam2->m_ColumnTexts[pListCtrl->m_nSortColumn];
 if(IsNumber(pszText1)) //按数值比较
 return pListCtrl->m_bAscend ? CompareDataAsNumber(pszText1, pszText2)
 : CompareDataAsNumber(pszText2, pszText1);
 else //按文本比较
 return pListCtrl->m_bAscend ? lstrcmp(pszText1, pszText2)
 : lstrcmp(pszText2, pszText1);
}

```

## 举一反三

根据本实例, 读者可以:

- 在列表控件的列标题中显示图标。

## 实例 079

## 具有文本录入功能的列表视图控件

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\02\079

## 实例说明

列表视图控件简单易用, 但是不能进行编辑, 本实例将介绍如何使列表视图控件可编辑。



运行程序, 部门表的记录将显示在表格中, 在表格中可以对数据进行编辑; 单击“保存”按钮可以将数据保存到数据库中, 如图 2.32 所示。

### 技术要点

可以通过两种方法实现列表视图控件的可编辑功能, 一种是在要编辑的单元格位置创建一个编辑框控件, 另一种是创建一个编辑框控件, 并将该控件移动到要编辑的单元格所在的位置。

本实例使用的是第二种方法, 当用户单击表格中的单元格时, 将编辑框显示在单元格中, 用户可以在编辑框中输入数据, 在编辑框失去焦点时将数据写入单元格。

### 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 向窗体中添加一个列表视图控件。

(3) 以 CEdit 类为基类创建新类 CListEdit。

(4) 处理编辑框的 WM\_KILLFOCUS 消息, 使其在失去焦点时将数据显示在列表视图控件的单元格中, 代码如下:

```
void CListEdit::OnKillFocus(CWnd* pNewWnd)
{
 CGridList * temp = (CGridList *)GetParent(); //获得父窗口指针
 if(temp)
 {
 temp->DisposeEdit(); //设置编辑框控件
 }
}
```

(5) 从 CListCtrl 类中派生一个 CGridList 类, 改写 PreSubclassWindow 虚函数, 为列表视图控件设置风格, 创建编辑框, 代码如下:

```
void CGridList::PreSubclassWindow()
{
 // 修改列表视图控件风格
 ModifyStyle(LVS_EDITLABELS,0);
 ModifyStyle(0,LVS_REPORT);
 ModifyStyle(0,LVS_SHOWSELALWAYS);
 //设置列表视图控件扩展风格
 SetExtendedStyle(LVS_EX_FLATSB
 |LVS_EX_FULLROWSELECT
 |LVS_EX_HEADERDRAGDROP
 |LVS_EX_ONECLICKACTIVATE
 |LVS_EX_GRIDLINES);
 //创建编辑框控件
 edit.Create(WS_CHILD|WS_CLIPSIBLINGS|WS_EX_TOOLWINDOW|WS_BORDER,
 CRect(0,40,10,50),this,1001);
 CListCtrl::PreSubclassWindow();
}
```

(6) 根据鼠标单击时的位置确定编辑框应该出现的位置, 需要处理表格的 WM\_LBUTTONDOWN 消息, 以此来确定用户单击的单元格的坐标, 代码如下:

```
void CGridList::OnLButtonDown(UINT nFlags, CPoint point)
{
 CListCtrl::OnLButtonDown(nFlags, point);
 LVHITTESTINFO info;
 info.pt=point;
 info.flags=LVHT_ONITEMLABEL;
 if(SubItemHitTest(&info)>=0)
 {
 row=info.iItem;
 col=info.iSubItem;
 ShowEdit();
 }
}
```

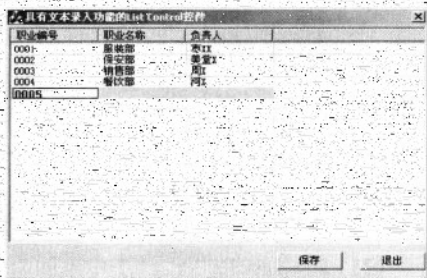


图 2.32 具有文本录入功能的 ListControl 控件

(7) 添加 ShowEdit 函数, 用于显示编辑框, 代码如下:

```
void CGridList::ShowEdit()
{
 CRect rect;
 GetSubItemRect(row,col,LVIR_LABEL,rect); //获得列表项区域
 CString str;
 str = GetItemText(row,col); //获得行列信息
 edit.MoveWindow(rect); //移动编辑框位置
 edit.SetWindowText(str); //设置编辑框文本
 edit.ShowWindow(SW_SHOW); //显示编辑框
 edit.SetSel(0,100); //设置编辑框中文本选中
 edit.SetFocus(); //设置编辑框焦点
 UpdateWindow(); //更新窗口
}
```

(8) 添加 DisposeEdit 函数, 为列表视图控件的单元格赋值并隐藏编辑框, 代码如下:

```
BOOL CGridList::DisposeEdit()
{
 CString sLabel;
 edit.GetWindowText(sLabel); //获得编辑框文本
 this->SetItemText(row,col,sLabel); //插入到列表的指定单元格中
 edit.ShowWindow(SW_HIDE); //隐藏编辑框
 ::SendMessage(this->GetParent()->GetSafeHwnd(),NULL,NULL,NULL);
 return true;
}
```

(9) 为“保存”按钮添加消息响应函数, 把表格中的数据添加到数据库中, 代码如下:

```
void CTextInListDlg::OnBtsSave()
{
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn(); //连接数据库
 CString sql,str1,str2;
 sql.Format("delete from bumenbiao"); //设置删除语句
 m_AdoConn.ExecuteSQL((_bstr_t)sql); //执行删除语句
 int m=0;
 for(int i=0;i<m_Grid.GetItemCount();i++)
 {
 //获得列表中数据
 str=m_Grid.GetItemText(i,m);
 str1=m_Grid.GetItemText(i,m+1);
 str2=m_Grid.GetItemText(i,m+2);
 if(!str.IsEmpty() || !str1.IsEmpty() || !str2.IsEmpty())
 {
 //将列表中数据插入到数据库中
 sql.Format("insert into bumenbiao (职业编号,职业名称,负责人) values ('%s','%s','%s')",str,str1,str2);
 m_AdoConn.ExecuteSQL((_bstr_t)sql); //执行插入语句
 }
 }
 m_AdoConn.ExitConnect();
 MessageBox("保存完毕");
}
```

### 举一反三

根据本实例, 读者可以:

- 实现联想录入功能。

## 实例 080

## 使用列表视图设计登录界面

这是一个可以启发思维的实例

实例位置: 光盘\mingrisoft\02\080

### 实例说明

用户在登录一些软件时, 经常可以看到用户名是以图标的形式显示在列表中的。本实例就是使用列表视图设计登录界面, 实例运行结果如图 2.33 所示。

### 技术要点

首先创建一个图像列表, 并通过 SetImageList 方法将列表视图控件和图像列表关联到一起。语法如下:



图 2.33 使用列表视图设计登录界面



CImageList\* SetImageList( CImageList\* pImageList, int nImageList );

参数说明:

- pImageList: 标识图像列表指针。
- nImageList: 标识图像列表类型, 可选值如下。
  - LVSIL\_NORMAL: 图像列表具有大图标。
  - LVSIL\_SMALL: 图像列表具有小图标。
  - LVSIL\_STATE: 图像列表具有状态图像。

然后调用 InsertItem 方法向列表视图控件插入数据。语法如下:

```
int InsertItem(int nItem, LPCTSTR lpszItem);
int InsertItem(int nItem, LPCTSTR lpszItem, int nImage);
int InsertItem(UINT nMask, int nItem, LPCTSTR lpszItem, UINT nState, UINT nStateMask, int nImage, LPARAM lParam);
```

参数说明:

- pItem: 是 LVITEM 结构指针, LVITEM 结构中包含的视图项的文本、图像索引、状态等信息。
- nItem: 表示被插入的视图项索引。
- lpszItem: 表示视图项文本。
- nImage: 表示视图项图像索引。
- nMask: 一组标记, 用于确定哪一项信息是合法的。
- nState: 表示视图项的状态。
- nStateMask: 确定设置视图项的哪些状态。
- lParam: 表示关联视图项的附加信息。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向工程中导入 7 个图标资源。
- (3) 向对话框中添加一个列表视图控件、一个静态文本控件、一个编辑框控件和一个按钮控件。
- (4) 在对话框头文件中声明一个图像列表对象 m\_ImageList。
- (5) 主要程序代码如下:

```
BOOL CLoginDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 // ...系统代码省略

 m_ImageList.Create(32,32,ILC_COLOR24|ILC_MASK,1,0); //创建列表视图窗口
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON1)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON2)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON3)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON4)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON5)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON6)); //向图像列表中添加图标
 m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON7)); //向图像列表中添加图标
 m_Icon.SetImageList(&m_ImageList,LVSIL_NORMAL); //将图像列表关联到列表视图控件中
 m_Icon.InsertItem(0,"王",0); //向列表视图中添加数据
 m_Icon.InsertItem(1,"孙",1); //向列表视图中添加数据
 m_Icon.InsertItem(2,"刘",2); //向列表视图中添加数据
 m_Icon.InsertItem(3,"吕",3); //向列表视图中添加数据
 m_Icon.InsertItem(4,"庞",4); //向列表视图中添加数据
 m_Icon.InsertItem(5,"宋",5); //向列表视图中添加数据
 m_Icon.InsertItem(6,"孙",6); //向列表视图中添加数据
 return TRUE;
}
```

## 举一反三

根据本实例, 读者可以:



● 设计腾讯 OICQ 抽屉界面。

## 2.7 树视图控件典型实例

树视图控件可以用来显示具有层次结构的数据，例如组织树、索引项、磁盘中的文件或成等级结构的数据库。下面通过几个实例介绍树视图控件的应用。

### 实例 081

#### 多级数据库树状结构数 据显示

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\02\081

#### 实例说明

树视图控件最大的特点就是可以非常清晰地显示数据之间的层次关系。本实例向大家介绍如何利用树状结构显示数据信息。运行程序，单击节点前的 $\oplus$ ，展开下一级节点，如图 2.34 所示。

#### 技术要点

在使用树视图控件时，需要通过 SetImageList 函数来关联图像列表，使用 InsertItem 函数向树型控件中插入数据，利用 GetNextItem 函数遍历数据。

(1) InsertItem 函数。InsertItem 函数用于插入节点。语法如下：

```
HTREEITEM InsertItem(LPTVINSERTSTRUCT lpInsertStruct);
HTREEITEM InsertItem(UINT nMask, LPCTSTR lpszItem, int nImage, int nSelectedImage,
 UINT nState, UINT nStateMask, LPARAM lParam, HTREEITEM hParent, HTREEITEM hInsertAfter);
HTREEITEM InsertItem(LPCTSTR lpszItem, HTREEITEM hParent = TVI_ROOT,
 HTREEITEM hInsertAfter = TVI_LAST);
HTREEITEM InsertItem(LPCTSTR lpszItem, int nImage, int nSelectedImage,
 HTREEITEM hParent = TVI_ROOT, HTREEITEM hInsertAfter = TVI_LAST);
```

参数说明：

- lpInsertStruct：是 TVINSERTSTRUCT 结构指针，TVINSERTSTRUCT 结构中包含了插入操作的详细信息。
- nMask：标识节点的哪些信息被设置。
- lpszItem：确定节点的文本。
- nImage：确定节点的图像索引。
- nSelectedImage：确定节点选中时的图像索引。
- nState：确定节点的状态。
- nStateMask：确定节点的哪些状态被设置。
- lParam：用于指定关联节点的附加信息。
- hParent：标识父节点句柄。
- hInsertAfter：标识新插入节点后面的节点句柄。

(2) GetNextItem 函数。GetNextItem 函数根据当前节点获取下一个节点。语法如下：

```
HTREEITEM GetNextItem(HTREEITEM hItem, UINT nCode);
```

参数说明：

- hItem：当前节点句柄。
- nCode：标识如何查找下一个节点，可选值如表 2.2 所示。



图 2.34 多级数据库树状结构数据显示





表 2.2

nCode 参数可选值表

| 可 选 值                | 描 述         |
|----------------------|-------------|
| TVGN_CARET           | 返回当前选中的节点   |
| TVGN_CHILD           | 返回第一个子节点    |
| TVGN_DROPHILITE      | 返回拖动的节点     |
| TVGN_FIRSTVISIBLE    | 返回第一个可见的节点  |
| TVGN_NEXT            | 返回下一个兄弟节点   |
| TVGN_NEXTVISIBLE     | 返回下一个可见节点   |
| TVGN_PARENT          | 返回所标识节点的父节点 |
| TVGN_PREVIOUS        | 返回上一个兄弟节点   |
| TVGN_PREVIOUSVISIBLE | 返回上一个可见节点   |
| TVGN_ROOT            | 返回根节点       |

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向工程导入 4 个图标资源，并向对话框中添加 1 个树视图控件。
- (3) 主要程序代码如下：

```
//定义图像列表，设置根节点
Load_dep(); //自定义函数，取出数据库中的数据
m_treeImageList.Create(16,16,ILC_MASK,4,1); //创建图像列表
m_treeImageList.Add(theApp.LoadIcon(IDI_ICON1)); //加载图标资源
m_treeImageList.Add(theApp.LoadIcon(IDI_ICON2));
m_treeImageList.Add(theApp.LoadIcon(IDI_ICON3));
m_treeImageList.Add(theApp.LoadIcon(IDI_ICON4));
HICON hIcon=::LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDR_MAINFRAME));
m_tree.SetImageList(&m_treeImageList,LVSIL_NORMAL); //关联图像列表
m_root=m_tree.InsertItem("国家",0,0); //插入根节点
AddToTree(m_root,0);
m_tree.Expand(m_root,TVE_EXPAND); //展开节点
//向树视图控件插入数据
void CDJTreeDlg::AddToTree(HTREEITEM m_node,int UPID)
{
 HTREEITEM m_child;
 for(int i=0;i<a_upid.GetSize();i++)
 {
 if(UPID==atoi(a_upid.GetAt(i)))
 {
 switch(atoi(a_lx.GetAt(i)))
 {
 case 1:
 m_child = m_tree.InsertItem(a_name.GetAt(i),2,2,m_node);
 AddToTree(m_child,atoi(a_id.GetAt(i)));
 break;
 case 2:
 m_child = m_tree.InsertItem(a_name.GetAt(i),1,1,m_node);
 AddToTree(m_child,atoi(a_id.GetAt(i)));
 break;
 case 3:
 m_child = m_tree.InsertItem(a_name.GetAt(i),3,3,m_node);
 AddToTree(m_child,atoi(a_id.GetAt(i)));
 break;
 }
 }
 }
}
```

## 举一反三

根据本实例，读者可以：

- 实现学生分级管理。

## 实例 082 节点拖动功能的树控件

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\02\082

## 实例说明

在程序中使用树控件通常描述一些具有层次关系的数据。例如，使用树控件显示企业的人事信息，显示地域信息等。有时，由于一些原因这些数据的层次关系会发生改变，如果能够利用鼠标拖动树控件中的节点来修改数据的层次关系，会极大地简化用户操作流程。运行程序，拖动树控件节点，如图 2.35 所示。

## 技术要点

在树控件中实现节点的拖动，首先需要调用 `CreateDragImage` 方法创建一个拖动的图像列表 (`CImageList`)，然后调用图像列表的 `BeginDrag` 方法开始一个拖曳操作，调用图像列表的 `DragEnter` 方法锁定窗口更新，在拖动过程中显示图像。接着在鼠标移动的过程中调用图像列表的 `DragMove` 方法移动图像，显示拖动的效果。最后调用图像列表的 `DragLeave` 方法解除对窗口的锁定，隐藏拖动的图像，调用图像列表的 `EndDrag` 方法结束拖曳操作。下面介绍这些方法的使用。



图 2.35 节点拖动功能的树控件

(1) `CreateDragImage` 方法。树控件的 `CreateDragImage` 方法为指定的节点创建一个拖动的位图，并且为该位图创建一个图像列表，将位图添加到图像列表中。语法如下：

```
CImageList* CreateDragImage(HTREEITEM hItem);
```

参数说明：

- `hItem`：表示树控件中指定的节点。

返回值：方法返回值是图像列表指针。

(2) `BeginDrag` 方法。图像列表 `CImageList` 的 `BeginDrag` 方法用于开始拖动一个图像。语法如下：

```
BOOL BeginDrag(int nIndex, CPoint ptHotSpot);
```

参数说明：

- `nIndex`：表示拖动的图像索引。
- `ptHotSpot`：表示开始拖动的起点坐标。

返回值：如果方法执行成功，返回值为 `TRUE`，否则为 `FALSE`。

(3) `DragEnter` 方法。图像列表的 `DragEnter` 方法用于在拖动过程中锁定窗口更新，显示拖动的图像。语法如下：

```
static BOOL PASCAL DragEnter(CWnd* pWndLock, CPoint point);
```

参数说明：

- `pWndLock`：表示需要锁定的窗口对象。
- `point`：表示显示拖动图像的位置。

返回值：如果方法执行成功，返回值为 `TRUE`，否则为 `FALSE`。

(4) `DragMove` 方法。图像列表的 `DragMove` 方法用于在拖动过程中移动图像到指定的位置。语法如下：

```
static BOOL PASCAL DragMove(CPoint pt);
```

参数说明：

- `pt`：表示将图像移动到新的位置。

返回值: 如果方法执行成功, 返回值为 TRUE, 否则为 FALSE。

(5) DragLeave 方法。图像列表的 DragLeave 方法用于解除对窗口的更新, 隐藏拖动的图像。语法如下:

```
static BOOL PASCAL DragLeave(CWnd* pWndLock);
```

参数说明:

● pWndLock: 表示之前锁定更新的窗口对象。

返回值: 如果方法执行成功, 返回值为 TRUE, 否则为 FALSE。

(6) EndDrag 方法。图像列表的 EndDrag 方法用于结束拖拽操作。语法如下:

```
static void PASCAL EndDrag();
```

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 向对话框中添加树控件, 设置树控件的属性如图 2.36 所示。

(3) 从 CTreeCtrl 派生一个子类 CDragTree, 利用类向导将树控件命名为“m\_DragTree”, 其类型为“CDragTree”。

(4) 向 CDragTree 类中添加成员变量。代码如下:

```
CImageList* m_pDragImages; //拖动的图像列表
BOOL m_bDrag; //是否进行拖动
HTREEITEM m_hBeginDrag; //拖动的起点
```

(5) 向 CDragTree 类中添加 CopyNodes 方法, 在拖动节点时, 进行节点的移动, 将源节点及其子节点复制到目标节点下。代码如下:

```
void CDragTree::CopyNodes(HTREEITEM hDesItem, HTREEITEM hSrcItem)
{
 if (hDesItem==NULL || hSrcItem==NULL) //验证参数
 {
 return;
 }
 TVITEM tvItem; //定义项目信息
 tvItem.mask = TVIF_TEXT|TVIF_IMAGE; //设置返回标记
 tvItem.hItem = hSrcItem;
 char chText[MAX_PATH] = {0};
 tvItem.pszText = chText;
 tvItem.cchTextMax = MAX_PATH;
 GetItem(&tvItem); //获取项目信息
 TVINSERTSTRUCT tvInsert; //定义插入操作的数据结构
 tvInsert.hParent = hDesItem;
 tvInsert.item = tvItem;
 HTREEITEM hInsert = InsertItem(&tvInsert); //插入项目
 HTREEITEM hChild = GetChildItem(hSrcItem); //获取子节点
 while (hChild != NULL) //遍历子节点
 {
 tvItem.mask = TVIF_TEXT|TVIF_IMAGE;
 tvItem.hItem = hChild;
 tvItem.pszText = chText;
 tvItem.cchTextMax = MAX_PATH;
 GetItem(&tvItem);
 tvInsert.hParent = hInsert;
 tvInsert.item = tvItem;
 CopyNodes(hInsert, hChild); //递归调用
 hChild = GetNextSiblingItem(hChild); //查找下一个兄弟节点
 }
}
```

(6) 向 CDragTree 类中添加 OnBeginDrag 方法, 在开始拖动节点时调用。代码如下:

```
void CDragTree::OnBeginDrag(NMHDR* pNMHDR, LRESULT* pResult)
{
 NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
 HTREEITEM hItem = pNMTreeView->itemNew.hItem; //获取开始拖动的节点
 if (hItem==GetRootItem()) //不允许拖动根节点
 {
 }
```

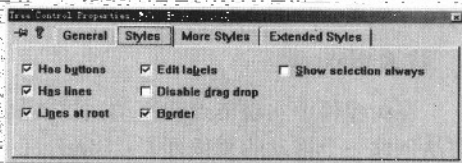


图 2.36 树控件属性设置

```

 *pResult = 0;
 return;
 }
 m_hBeginDrag = hItem;
 m_pDragImages = CreateDragImage(hItem); //记录开始拖动的项目
 CPoint dragPT; //创建拖动的图像列表
 dragPT.x = pNMTreeView->ptDrag.x; //记录起始点
 dragPT.y = pNMTreeView->ptDrag.y;
 if (m_pDragImages != NULL)
 {
 m_pDragImages->BeginDrag(0, CPoint(8, 8)); //开始拖动图像
 ClientToScreen(&dragPT); //转换客户坐标到屏幕坐标
 m_pDragImages->DragEnter(this, dragPT); //锁定窗口更新, 在拖动的过程中显示拖动的图像
 SetCapture(); //开始鼠标捕捉
 m_bDrag = TRUE;
 }
 *pResult = 0;
}

```

(7) 在 CDragTree 类的消息映射部分添加映射宏, 在用户拖动节点时执行 OnBeginDrag 方法。代码如下:

```
ON_NOTIFY_REFLECT(TVN_BEGINDRAG, OnBeginDrag)
```

(8) 在鼠标在树控件上移动时, 如果当前拖动节点, 则设置拖动图像的位置。代码如下:

```

void CDragTree::OnMouseMove(UINT nFlags, CPoint point)
{
 if (m_bDrag) //处于拖动状态
 {
 HTREEITEM hItem;
 UINT nHitFlags;
 CRect clientRC;
 GetClientRect(&clientRC);
 m_pDragImages->DragMove(point); //获取客户区域
 //设置拖动的图像位置
 //鼠标经过时高亮显示
 if (hItem = HitTest(point, &nHitFlags)) != NULL)
 {
 CImageList::DragShowNolock(FALSE); //隐藏拖动的图像
 SelectDropTarget(hItem); //设置选中的项目
 CImageList::DragShowNolock(TRUE); //显示拖动的图像
 }
 }
 else
 CTreeCtrl::OnMouseMove(nFlags, point);
}

```

(9) 处理释放鼠标按钮的单击事件, 如果用户正在进行拖动操作, 此时将结束拖动操作。代码如下:

```

void CDragTree::OnLButtonUp(UINT nFlags, CPoint point)
{
 if (m_bDrag) //处于拖动状态
 {
 m_bDrag = FALSE;
 CImageList::DragLeave(this); //解除对树控件的锁定, 隐藏拖动的图像
 CImageList::EndDrag(); //结束图像拖动
 ReleaseCapture(); //释放鼠标捕捉
 delete m_pDragImages; //释放图像列表
 m_pDragImages = NULL;
 CRect winRC;
 GetWindowRect(&winRC); //获取窗口区域
 HTREEITEM hItem;
 if (hItem = HitTest(point, &nFlags)) != NULL)
 {
 //进行拖动处理
 //如果目标项目与开始拖动的项目相同或者目标项目仍是开始项目的父节点, 不进行处理
 if (m_hBeginDrag != hItem && hItem != GetParentItem(m_hBeginDrag))
 {
 CopyNodes(hItem, m_hBeginDrag); //进行节点的复制
 DeleteItem(m_hBeginDrag); //删除源节点
 }
 Invalidate();
 SelectDropTarget(NULL);
 m_hBeginDrag = NULL;
 }
 }
}

```



## 举一反三

根据本实例,读者可以:

- 动态修改树控件节点。

## 实例 083 带复选功能的树状结构

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\02\083

## 实例说明

在树状结构的节点前添加复选框,可以使用户很方便地对相关信息进行检索。运行程序,选择节点前的复选框,单击“确定”按钮,在 RichEdit 控件中显示选择的节点信息,如图 2.37 所示。

## 技术要点

本实例利用了树视图控件的 Check Boxes 属性,在树状节点前添加复选框,然后使用 GetCheck 函数判断复选框是否被选中。GetCheck 函数原型如下:

```
BOOL GetCheck(HTREEITEM hItem);
```

参数说明:

- hItem: 节点句柄。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加一个树视图控件和一个 RichEdit 控件,用鼠标右键单击树视图控件,在弹出的快捷菜单中选择 Properties 选项,选择 Check Boxes 属性。

(3) 主要程序代码如下:

```
void CFXtreeDlg::OnOK()
{
 HTREEITEM item;
 item = m_tree.GetRootItem(); //获得根节点
 str="";
 if(m_tree.GetCheck(item)!=0) //如果根节点被选中
 {
 str+=m_tree.GetItemText(item); //获得根节点文本
 str+="\n";
 m_rich.SetWindowText(str);
 }
 OuttoTree(item);
 //CDialog::OnOK();
}

//递归判断节点前复选框是否选中
void CFXtreeDlg::OuttoTree(HTREEITEM m_node)
{
 HTREEITEM m_child;
 m_node = m_tree.GetChildItem(m_node); //获得子节点
 if (m_node != NULL)
 {
 while (m_node!= NULL)
 {
 if(m_tree.GetCheck(m_node)!=0) //判断当前节点是否选中
 {
 str+=m_tree.GetItemText(m_node);
 str+="\n";
 m_rich.SetWindowText(str);
 }
 m_child = m_node;
```

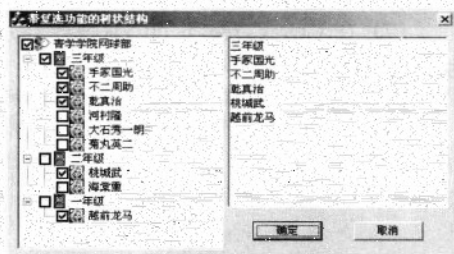


图 2.37 带复选功能的树状结构

```

 OuttoTree(m_child);
 m_node = m_tree.GetNextItem(m_node, TVGN_NEXT); //查找下一个节点
}
}

```

## 举一反三

根据本实例，读者可以：

- 实现在餐饮管理系统中显示某一餐台或包房的用餐信息。

## 实例 084 三态效果树控件

本实例可以方便操作、提高效率

实例位置：光盘\mingrisoft\02\084

## 实例说明

许多应用软件的树形控件具有漂亮的复选框，这是如何实现的呢？本实例利用位图设计了一个具有三态效果复选框的树形控件，如图 2.38 所示。

## 技术要点

实际上，树形控件中的复选框是通过图像状态索引绘制的。首先设计一个位图，其中包含视图项的各个状态，将位图添加到 CImageList 中，然后调用 CTreeCtrl 的 SetImageList 方法设置图像状态索引。



图 2.38 三态效果树控件

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加一个树视图控件，用鼠标右键单击树视图控件，在弹出的快捷菜单中选择 Properties 选项，为树控件设置属性。
- (3) 主要程序代码如下：

```

void CBitTreeCtrl::OnLButtonDown(UINT nFlags, CPoint point)
{
 HTREEITEM hItemInfo = HitTest(point, &m_Flags);
 nFlags = m_Flags;
 //TVHT_ONITEMSTATEICON表示用户定义的视图项的图标状态
 if (m_Flags & TVHT_ONITEMSTATEICON)
 {
 //State: 0无状态, 1没有选择, 2部分选择, 3全部选择
 //12~15位表示视图项的图像状态索引
 UINT State = GetItemState(hItemInfo, TVIS_STATEIMAGEMASK) >> 12;
 State = (State - 1) & 3;
 SetItemState(hItemInfo, INDEXTOSTATEIMAGEMASK(State), TVIS_STATEIMAGEMASK);
 }
 else
 CTreeCtrl::OnLButtonDown(nFlags, point);
}

//设置节点状态
BOOL CBitTreeCtrl::SetItemState(HTREEITEM hItem, UINT State, UINT StateMask, BOOL IsSearch)
{
 BOOL ret = CTreeCtrl::SetItemState(hItem, State, StateMask);

 UINT nState = State >> 12;
 if (nState != 0)
 {
 if (IsSearch)
 RansackChild(hItem, nState);
 RansackParentAndChild(hItem, nState);
 }
 return ret;
}

```

```

}

//遍历子节点
void CBitTreeCtrl::RansackChild(HTREEITEM hItem, UINT State)
{
 HTREEITEM hChildItem, hBrotherItem;
 //查找子节点
 hChildItem = GetChildItem(hItem);
 if(hChildItem != NULL)
 {
 //将所有子节点的状态设置与父节点相同
 CTreeCtrl::SetItemState(hChildItem, INDEXTOSTATEIMAGEMASK(State), TVIS_STATEIMAGEMASK);
 //再递归处理子节点
 RansackChild(hChildItem, State);

 //搜索子节点的兄弟节点
 hBrotherItem = GetNextSiblingItem(hChildItem);
 while (hBrotherItem)
 {
 //设置子节点的兄弟节点状态与当前节点的状态一致
 int nState = GetItemState(hBrotherItem, TVIS_STATEIMAGEMASK) >> 12;
 if(nState != 0)
 {
 CTreeCtrl::SetItemState(hBrotherItem, INDEXTOSTATEIMAGEMASK(State),
 TVIS_STATEIMAGEMASK);
 }
 //再递归处理兄弟节点
 RansackChild(hBrotherItem, State);
 hBrotherItem = GetNextSiblingItem(hBrotherItem);
 }
 }
}

void CBitTreeCtrl::RansackParentAndChild(HTREEITEM hItem, UINT State)
{
 HTREEITEM hNextItem, hPrevItem, hParentItem;
 //查找父节点, 没有就结束
 hParentItem = GetParentItem(hItem);
 if(hParentItem != NULL)
 {
 UINT nState1 = State; //设初始值, 防止没有兄弟节点时出错
 //查找当前节点的所有兄弟节点, 如果所有兄弟节点状态都相同
 //设置父节点的状态
 //查找当前节点下面的兄弟节点的状态
 hNextItem = GetNextSiblingItem(hItem);
 while(hNextItem != NULL)
 {
 nState1 = GetItemState(hNextItem, TVIS_STATEIMAGEMASK) >> 12;
 if(nState1 != State && nState1 != 0) break;
 else hNextItem = GetNextSiblingItem(hNextItem);
 }

 if(nState1 == State)
 {
 //查找当前节点上面的兄弟节点的状态
 hPrevItem = GetPrevSiblingItem(hItem);
 while(hPrevItem != NULL)
 {
 nState1 = GetItemState(hPrevItem, TVIS_STATEIMAGEMASK) >> 12;
 if(nState1 != State && nState1 != 0) break;
 else hPrevItem = GetPrevSiblingItem
 (hPrevItem);
 }
 }

 if(nState1 == State || nState1 == 0)
 {
 nState1 = GetItemState(hParentItem, TVIS_STATEIMAGEMASK) >> 12;
 if(nState1 != 0)
 {
 //如果状态一致, 则父节点的状态与当前节点的状态一致
 CTreeCtrl::SetItemState(hParentItem, INDEXTOSTATEIMAGEMASK(State),
 TVIS_STATEIMAGEMASK);
 }
 //再递归处理父节点的兄弟节点和其父节点
 RansackParentAndChild(hParentItem, State);
 }
 else
 {

```



```

//状态不一致,则当前节点的父节点、父节点的父节点……状态均为第三态
hParentItem=GetParentItem(hItem);
while(hParentItem!=NULL)
{
 nState1 = GetItemState(hParentItem, TVIS_STATEIMAGEMASK) >> 12;
 if(nState1!=0)
 {
 CTreeCtrl::SetItemState(hParentItem, INDEXTOSTATEIMAGEMASK(2),
 TVIS_STATEIMAGEMASK);
 }
 hParentItem=GetParentItem(hParentItem);
}
}
}
}

```

## 举一反三

根据本实例,读者可以:

- 自绘树控件。

## 实例 085 修改树控件节点连线颜色

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\085

## 实例说明

在使用一些软件时,经常可以看到漂亮的树控件,而且这些树控件的节点连线都不是默认的黑色,本实例就实现了修改树控件的节点连线颜色的功能。程序运行结果如图 2.39 所示。

## 技术要点

树控件默认的节点连线、加号减号标记均是黑色的;为了修改默认的黑色,需要将树控件发送一个消息。在 Visual C++ 2005 中,树控件提供了 SetLineColor 方法用于修改节点连线的颜色,但是在 Visual C++ 6.0 中没有提供该访问,只能采用原始发送消息的方式实现。发送的消息值为 TV\_FIR ST + 40,其中该消息的第 2 个参数 lParam 表示的是设置的颜色。下面的代码演示了设置节点连线的颜色。

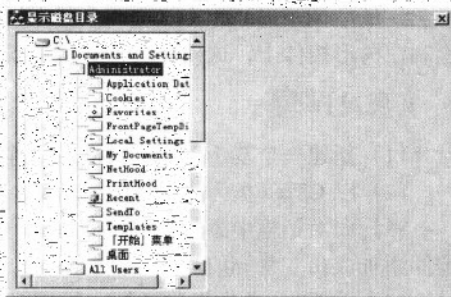


图 2.39 修改树控件节点连线颜色

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 以 CTree 类为基类派生一个 CLineTree 类。
- (3) 在对话框上添加树视图控件,设置控件属性,并添加 CLineTree 类的成员变量 m\_Tree。
- (4) 主要程序代码如下:

```

CLineTree::CLineTree()
{
 m_clText = RGB(58, 79, 107);
 m_clBk = RGB(234, 242, 255);
 m_clLine = RGB(0, 0, 255);
}

void CLineTree::PreSubclassWindow()
{
 //设置文本颜色
 SetTextColor(m_clText);
 //设置背景颜色
 SetBkColor(m_clBk);
}

```



```
//设置线条颜色
SendMessage(TV_FIRST + 40, 0, (LPARAM)m_clLine);
CTreeCtrl::PreSubclassWindow();
}
}
```

## 举一反三

根据本实例，读者可以：

- 绘制树控件的背景及文本颜色。

## 实例 086 位图背景树控件

这是一个可以启发思维的实例

实例位置：光盘\mingrisoft\02\086

### 实例说明

用户在使用一些软件时，有时会看到界面中的树状结构具有位图背景。本实例就设计了具有位图背景的树控件，实例运行结果如图 2.40 所示。

### 技术要点

为了实现具有背景位图的树控件，首先要获得树形控件原始图像，并将原始图像绘制在一个画布对象上，然后再定义一个画布对象，将背景图片绘制在该画布对象上，最后调用 BitBlt 方法将两个画布对象进行与运算绘制在树形控件上，这样完成了具有背景位图的树控件的设计。

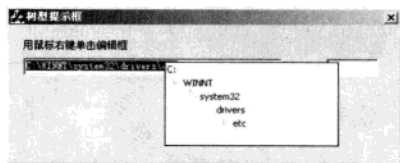


图 2.40 位图背景树控件

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 以 CTree 类为基类派生一个 CBitmapTree 类。
- (3) 向对话框中添加一个树控件、一个静态文本控件和一个按钮控，为树控件和静态文本控件添加成员变量 m\_Tree 和 m\_Source。
- (4) 以 CDC 类为基类派生一个 CMemDC 类，代码如下：

```
class CMemDC : public CDC
{
private:
 CBitmap* m_Bmp; //位图对象指针
 CBitmap* m_OldBmp; //位图对象指针
 CDC* m_pDC; //设备上下文
 CRect m_Rect; // CRect对象
public:
 CMemDC(CDC* pDC, const CRect& rect) : CDC() //构造函数
 {
 CreateCompatibleDC(pDC); //创建与内存兼容的设备上下文
 m_Bmp = new CBitmap; //初始化内存兼容位图
 m_Bmp->CreateCompatibleBitmap(pDC, rect.Width(), rect.Height());
 m_OldBmp = SelectObject(m_Bmp); //调入位图
 m_pDC = pDC; //获得设备上下文
 m_Rect = rect; //获得矩形区域
 }
 ~CMemDC() //析构函数
 {
 m_pDC->BitBlt(m_Rect.left, m_Rect.top, m_Rect.Width(), m_Rect.Height(),
 this, m_Rect.left, m_Rect.top, SRCCOPY); //绘制位图
 SelectObject(m_OldBmp);
 if(m_Bmp != NULL)
 delete m_Bmp;
 }
};
```

(5) 在 CBitmapTree 类的头文件中声明变量，代码如下：

```
CBitmap m_Bitmap; //位图对象
HANDLE m_hBmp; //保存位图资源句柄
CString m_Path; //保存位图路径
```

(6) 在 CBitmapTree 类中，处理按钮的 WM\_PAINT 事件，在该事件的处理函数中为树控件绘制位图背景，代码如下：

```
void CBitmapTree::OnPaint()
{
 CPaintDC dc(this);
 m_hBmp = LoadImage(NULL,m_Path,IMAGE_BITMAP,0,0,LR_LOADFROMFILE); //加载位图资源
 CRect rect;
 GetClientRect(&rect); //获得客户区域
 m_Bitmap.Attach(m_hBmp);
 CDC memdc;
 memdc.CreateCompatibleDC(&dc); //创建内存兼容设备上下文
 CBitmap bitmap;
 bitmap.CreateCompatibleBitmap(&dc, rect.Width(), rect.Height()); //初始化位图
 memdc.SelectObject(&bitmap); //选入位图
 CWnd::DefWindowProc(WM_PAINT, (WPARAM)memdc.m_hDC, 0); //获取原始画布
 CMemDC tempDC(&dc,rect); //构造CMemDC对象
 CBrush brush;
 brush.CreatePatternBrush(&m_Bitmap); //创建位图画刷
 tempDC.FillRect(rect, &brush); //用位图画刷填充客户区域
 //将原始图片与背景进行组合
 tempDC.BitBlt(rect.left, rect.top, rect.Width(), rect.Height(),&memdc, rect.left, rect.top,SRCAAND);
 brush.DeleteObject(); //删除画刷对象
 m_Bitmap.Detach(); //分离位图对象
}
```

(7) 在 CBitmapTree 类中，添加自定义函数 SetTreeBK，该函数用于获得位图资源路径，代码如下：

```
BOOL CBitmapTree::SetTreeBK(CString path)
{
 m_Path = path;
 Invalidate();
 return true;
}
```

(8) 在 CBitmapTree 类中，处理按钮的 TVN\_ITEMEXPANDING 事件，在该事件的处理函数中重绘树控件背景，代码如下：

```
void CBitmapTree::OnItemexpanding(NMHDR* pNMHDR, LRESULT* pResult)
{
 NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
 SetRedraw(FALSE); //重绘树控件背景
 *pResult = 0;
}
```

(9) 在 CBitmapTree 类中，处理按钮的 TVN\_ITEMEXPANDING 事件，在该事件的处理函数中禁止重绘树控件背景，代码如下：

```
void CBitmapTree::OnItemexpanded(NMHDR* pNMHDR, LRESULT* pResult)
{
 NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
 SetRedraw(); //不进行重绘
 *pResult = 0;
}
```

### 举一反三

根据本实例，读者可以：

- 在树型控件中显示树型提示框。

## 实例 087 显示磁盘目录

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\02\087

### 实例说明

本实例实现用树型控件显示磁盘目录，运行程序，在树型控件中将显示磁盘的分区，通过

双击树型控件的节点可以查看该节点下的子目录。

程序运行结果如图 2.41 所示。

## 技术要点

本实例主要通过 GetLogicalDriveStrings 函数先获取磁盘分区, 然后处理树型控件的双击消息, 双击某个目录就是用循环查找该目录下的全部子目录。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加树视图控件, 设置 ID 属性为 IDC\_TRDISKTREE, 添加成员变量 m\_trdisktree。

(3) OnInitDialog 方法中完成树状结构根节点的初始化, 代码如下:

```
BOOL CDiskCataDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 imlst.Create(16,16,ILC_COLOR32|ILC_MASK,0,0); //创建图像列表
 m_trdisktree.SetImageList(&imlst,TVSIL_NORMAL); //关联图像列表
 m_trdisktree.ModifyStyle(0L,TVS_HASLINES|TVS_LINESATROOT); //修改控件属性
 size_t alldriver=::GetLogicalDriveStrings(0,NULL); //获取磁盘分区
 _TCHAR *driverstr;
 driverstr=new _TCHAR[alldriver+sizeof(_T(""))];
 if(GetLogicalDriveStrings(alldriver,driverstr)!=alldriver-1) //获得磁盘目录
 return FALSE;
 _TCHAR *pdriverstr=driverstr;
 size_t driversize=strlen(pdriverstr);
 HTREEITEM disktree;
 while(driversize>0)
 {
 SHGetFileInfo(pdriverstr,0,&fileinfo,sizeof(fileinfo),
 SHGFI_ICON); //获取系统文件图标
 imindex=imlst.Add(fileinfo.hIcon);
 disktree=m_trdisktree.InsertItem(pdriverstr,imindex,imindex,
 TVI_ROOT,TVI_LAST); //插入到树控件中
 pdriverstr+=driversize+1;
 driversize=strlen(pdriverstr);
 }
 return TRUE;
}
```

(4) 添加对 TVN\_SELCHANGED 消息的处理函数, 该函数实现在用户选择树状结构中某项时列出该项的子项, 代码如下:

```
void CDiskCataDlg::OnSelchangedTrdisktree(NMHDR* pNMHDR, LRESULT* pResult)
{
 NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
 CFileFind filefd;
 HTREEITEM parent;
 HTREEITEM rootitem=m_trdisktree.GetSelectedItem(); //获得当前选择的节点
 if(m_trdisktree.GetChildItem(rootitem))return; //判断是否有子节点
 parent=rootitem;
 CString rootstr=m_trdisktree.GetItemText(rootitem); //获得下一个节点
 CString temp;
 CString lstr;
 if(rootstr.Find("\\")==2)
 {
 lstr.Format("%s*.%",rootstr);
 }
 else
 {
 CString strparent;
 while(1)
 {
 parent=m_trdisktree.GetParentItem(parent);
 strparent=m_trdisktree.GetItemText(parent);
 }
 }
}
```

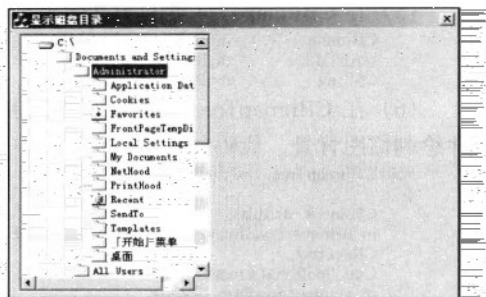


图 2.41 显示磁盘目录

```

if(strparent.Find("\\")==2)
 goto end;
temp+=strparent;
temp+="\\";
}
end:
CString root=m_trdisktree.GetItemText(parent);
lstr.Format("%s%s%s*.*",root,temp,rootstr);
}
BOOL bfinded=filefd.FindFile(lstr);
//循环插入数据
while(bfinde)
{
 bfinded=filefd.FindNextFile();
 CString filepath;
 if(filefd.IsDirectory()&&!filefd.IsDots()){
 SHGetFileInfo(filefd.GetFilePath(),0,&fileinfo,sizeof(fileinfo),
 SHGFI_ICON);
 imindex=imlst.Add(fileinfo.hIcon);
 m_trdisktree.InsertItem(filefd.GetFileName(),imindex,imindex,rootitem);
 }
}
*pResult = 0;
}

```

### 举一反三

根据本实例，读者可以：

- 利用树型控件显示全国行政区域。

## 实例 088 树型提示框

这是一个可以启发思维的实例

实例位置：光盘\mingrisoft\02\088

### 实例说明

树型提示框是将比较长的字符串用树状结构显示其层次效果，对字符串的要求是要有相同的字符作为分隔符，例如一个目录比较深的文件夹名。本实例实现了树型提示框的功能。运行程序，在文本框的字符上单击鼠标右键，提示窗体就会显示，实例运行结果如图 2.42 所示。

### 技术要点

本实例的实现需要新建基类为 CEdit 的新类 treetipedit，在新类中处理鼠标右键的消息，鼠标右键消息产生时就提取文本框中的字符，通过在长字符串中查找分隔符将其分成若干字符，然后将这若干字符按层次关系添加到树型控件中。

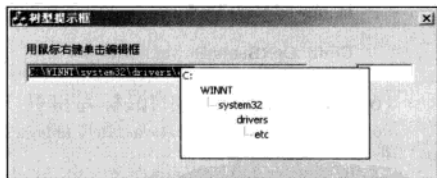


图 2.42 树型提示框

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在工程中添加基于 CTreeCtrl 的新类 tipwnd，添加基于 CEdit 的新类 treetipedit。
- (3) 在对话框上添加两个静态文本控件，设置 Caption 属性分别为“用鼠标右键单击编辑框”和“分隔符”；添加两个编辑框控件，设置 ID 属性分别为 IDC\_EDSTR 和 IDC\_EDSEP，添加 IDC\_EDSTR 的成员变量 m\_str，继承自类 treetipedit。

- (4) 在 treetipedit 类中添加显示树型窗体及窗体内容的 ShowTipWindow 函数，代码如下：

```

void tipwnd::ShowTipWindow(CString strtip,CString strsepar,CWnd* pParent,CPoint ptmouse)
{
 if (::IsWindow(m_hWnd))

```



```

return;
CRect rctWnd;
CWnd* pMain = CWnd::GetDesktopWindow();
pParent->GetWindowRect(&rctWnd); //获得窗口区域
pMain->ScreenToClient(&rctWnd); //转换为客户坐标

CPoint ptParent(ptmouse.x + rctWnd.left -3, ptmouse.y + rctWnd.top -3);
CRect rct(ptParent.x, ptParent.y, ptParent.x + 500, ptParent.y + 300);
Create(WS_BORDER|TVS_HASLINES|TVS_NOTOOLTIPS ,rct, pMain,-1); //创建树控件

HTREEITEM parent=TVI_ROOT;
CString strleft;
int pos=strlip.Find(strsepar); //查找分隔符
//按分隔符取出字符串插入到树结构中
if(pos<=0)return;
while(pos>0)
{
 strleft=stript.Left(pos);
 stript=stript.Right(stript.GetLength()-pos-strsepar.GetLength());
 parent=InsertItem(strleft,parent);

 pos=stript.Find(strsepar);
 if(pos<0)
 {
 parent=InsertItem(stript,parent);
 }
}
Expand(TVI_ROOT,TVE_EXPAND); //展开节点
SetItemState(parent,TVIS_SELECTED ,0); //设置节点状态
Select(parent,TVGN_CARET);

MoveWindow(rct.left,rct.top,200,100,FALSE); //移动树控件
SetFocus(); //设置焦点
SetWindowPos(&wndTopMost ,0, 0, 0, 0, SWP_NOMOVE | SWP_NOSIZE);
DWORD style = GetWindowLong(m_hWnd, GWL_EXSTYLE);
SetWindowLong(m_hWnd, GWL_EXSTYLE, style | WS_EX_TOPMOST); //设置风格
ShowWindow(SW_SHOW); //显示控件
}

```

(5) 在 `treetipedit` 类的鼠标右键按下事件中, 显示提示窗口, 代码如下:

```

void treetipedit::OnRButtonDown(UINT nFlags, CPoint point)
{
 CString edittext;
 GetWindowText(edittext); //获得提示字符串
 m_tipwnd.ShowTipWindow(edittext,strseparator,this,point); //显示提示窗口

 CEdit::OnRButtonDown(nFlags, point);
}

```

(6) 在 `treetipedit` 类的鼠标左键按下事件中, 关闭提示窗口, 代码如下:

```

void treetipedit::OnLButtonDown(UINT nFlags, CPoint point)
{
 m_tipwnd.ClostTipWinow(); //关闭提示窗口
 CEdit::OnLButtonDown(nFlags, point);
}

```

(7) 在 `OnInitDialog` 中设置编辑框中的字符串, 代码如下:

```

BOOL CTreeTipDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 //... 此处代码省略
 CString strtemp="\\\\";
 GetDlgItem(IDC_EDSEP)->SetWindowText(strtemp); //设置分隔符
 GetDlgItem(IDC_EDSTR)->SetWindowText("C:\\WINNT\\system32\\drivers\\etc"); //设置文本
 m_str.strseparator=strtemp;
 return TRUE;
}

```

## 举一反三

根据本实例, 读者可以:

- 在树型控件中显示树型提示框。

## 2.8 RichEdit 控件典型实例

### 实例 089

### 利用 RichEdit 显示 Word 文档

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\02\089

#### 实例说明

RichEdit 控件除了像编辑框一样显示基本数据外，还有多种用途，本实例将使用 RichEdit 控件显示 Word 文档中的内容，运行程序，单击“查找 Word 文档”按钮，在“打开”对话框中选择一个 Word 文档，单击“显示”按钮，文档中的内容将显示在 RichEdit 控件中，效果如图 2.43 所示。

#### 技术要点

要实现显示 Word 文档功能，首先需要通过类向导工程中导入几个 Word 类，分别是应用程序类 (\_Application)、文档管理对象 (Documents)、文档对象 (\_Document)、选区 (Range)；然后根据需要，对 RichEdit 控件的属性进行设置。

RichEdit 控件的主要属性如下。

- Multiline：能够显示多行文本，如果用户想要按 Enter 键在控件中换行，还需要控件具有 AutoHScroll 和 Want return 属性。
- Number：只允许输入数字。
- Horizontal scroll：为多行控件提供水平滚动条。
- Auto Hscroll：当用户在控件右方输入字符时，自动地向右方滚动文本。
- Vertical scroll：为多行文本控件提供垂直滚动条。
- Auto Vscroll：在多行文本控件中，当用户在最后一行按 Enter 键时，文本自动向上滚动。
- Password：以“\*”符号代替显示的文本。
- No hide selection：当失去或获得焦点时，不隐藏被选中的部分。
- OEM convert：能够转换 OEM 字符集。
- Want return：当用户的多行文本控件中按 Enter 键时，插入回车符。
- Border：具有边框。
- Uppercase：将所有字符转换为大写。
- Lowercase：将所有字符转换为小写。
- Read-only：文本是只读的。
- Left scrollbar：如果垂直滚动条被提供，它将显示在左边的客户区域。

#### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加一个 RichEdit 控件、一个编辑框控件，为 RichEdit 控件设置 Multiline、Horizontal scroll、Auto Hscroll、Vertical scroll、Want return、Border 等属性。
- (3) 在 InitInstance 函数中调用 AfxInitRichEdit 函数，用于初始化 RichEdit 控件。

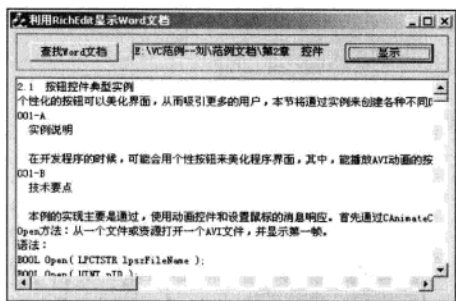


图 2.43 利用 RichEdit 显示 Word 文档

(4) 主要程序代码如下:

```
void CXSWordDlg::OnButxs()
{
 //Word应用程序
 Application app;
 //初始化连接
 app.CreateDispatch("word.Application");
 Documents doc;
 CComVariant a(_T(strText)),b(false),c(0),d(true);
 Document doc1;

 doc.AttachDispatch(app.GetDocuments());
 doc1.AttachDispatch(doc.Add(&a,&b,&c,&d));
 Range range;
 //求出文档的所选区域
 range = doc1.GetContent();//取出文件内容
 CString str;
 str = range.GetText();
 m_rich.SetWindowText(str);
 //关闭
 app.Quit(&b,&c,&c);
 //释放环境
 app.ReleaseDispatch();
}
```

### 举一反三

根据本实例,读者可以:

- 使用 RichEdit 控件显示文本文件。

## 实例 090

### 利用 RichEdit 控件实现 文字定位与标识

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\090

### 实例说明

文本编辑软件一般都有查找字符的功能,本实例实现了在 RichEdit 控件中查找字符,并将查找到的字符设为选中状态。运行程序,在文本框中输入想要查找的字符串,单击“查找”按钮,程序将在 RichEdit 控件中进行查找,再次单击“查找”按钮就会查找下一个相匹配的字符。实例运行结果如图 2.44 所示。

### 技术要点

本实例的实现主要通过 CRichEdit 类的 SetSel、LineFromChar 和 LineIndex 等方法,和 CString 类的 Find 方法实现。SetSel 方法是将控件中的字符设置为选中状态;LineFromChar 方法是根据字符在控件中的索引获得所在行的索引;LineIndex 方法用于获得字符在本行的索引数;Find 方法用来查找字符,并能返回字符的位置索引,根据这个位置索引可以计算出下一次开始查找的位置。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框上添加 RichEdit 控件,设置 ID 属性为 IDC\_RICHEDIT1,添加成员变量 m\_richedit;添加编辑框控件,设置 ID 属性为 IDC\_EDFIND,添加成员变量 m\_edfind;添加 Button 控件,设置 ID 属性为 IDC\_BTN\_FIND。
- (3) RichTextCharDlg.h 文件中加入变量声明:



图 2.44 利用 RichEdit 控件  
实现文字定位与标识

```
CString str;
CString tmp;
int istartpos;
int lineindex;
int movepos;
```

(4) 在 OnInitDialog 中实现文本文件的读取并显示在 RichEdit 控件中, 代码如下:

```
BOOL CRichTextCharDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 CStdioFile file;
 file.Open("test.txt",CFile::modeRead); //打开文件

 while(1)
 {
 DWORD i=file.ReadString(str); //读取文件中字符串
 if(i==0)goto end;
 tmp+=str;
 tmp+="\n";
 }
 end:m_richedit.SetWindowText(tmp); //设置控件显示文本
 lineindex=0;
 istartpos=0;
 movepos=0;
 return TRUE;
}
```

(5) 处理“查找”按钮的单击事件, 该事件的实现函数的代码如下:

```
void CRichTextCharDlg::OnFind()
{
 m_richedit.LineScroll(-lineindex);
 CString strfind;
 GetDlgItem(IDC_EDFIND)->GetWindowText(strfind); //获得待查找的字符串
 int ret=tmp.Find(strfind,istartpos); //查找字符串
 int strlen=strfind.GetLength(); //获得查找字符串长度
 m_richedit.SetSel(ret,ret+strlen); //选中的字符串
 istartpos=ret+strlen;
 lineindex=m_richedit.LineFromChar(ret);
 int linepos=m_richedit.LineIndex(lineindex);
 m_richedit.LineScroll(lineindex);
 m_richedit.SetFocus();
}
```

### 举一反三

根据本实例, 读者可以:

- 在 RichEdit 控件中标识所有匹配的字符。

### 实例 091

### 利用 RichEdit 控件显示 图文数据

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\02\091

### 实例说明

在对一些技术知识进行讲解时, 如果适当地插入一些图片, 将会为用户更好地掌握知识提供有力的帮助。运行程序, 用户直接在窗体中编辑文本, 在适当的位置定位光标, 单击“插入图片”按钮插入图片, 如图 2.45 所示。

### 技术要点

要实现图文显示功能, 首先需要使用 API 函数 LoadImage 装载图片, 然后创建并插入 OLE 对象。

LoadImage 函数装载图标、光标或位图。函数原



图 2.45 利用 RichEdit 控件显示图文数据





型如下:

HANDLE LoadImage( HINSTANCE hinst, LPCTSTR lpszName, UINT uType, int cxDesired, int cyDesired, UINT fuLoad );

参数说明:

- hinst: 处理包含被装载图像模块的特例。若要装载 OEM-图像, 则设此参数值为 0。
- lpszName: 处理图像装载。
- uType: 指定被装载图像类型。
- cxDesired: 指定图标或光标的宽度, 以像素为单位。
- cyDesired: 指定图标或光标的高度, 以像素为单位。
- fuLoad: 表示文件加载标识, 可选值如表 2.3 所示。

表 2.3

fuLoad 参数可选值表

| 可 选 值               | 描 述                                                                                                                             |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------|
| LR_DEFAULTCOLOR     | 默认标识, 它不作任何事情                                                                                                                   |
| LR_CREATEDIBSECTION | 当参数 uType 指定为 IMAGE_BITMAP 时, 使得函数返回一个 DIB 部分位图, 而不是一个兼容的位图                                                                     |
| LR_DEFAULTSIZE      | 若 cxDesired 或 cyDesired 未被设为零, 使用系统指定的公制值标识光标或图标的宽和高。如果这个参数不被设置且 cxDesired 或 cyDesired 被设为零, 函数使用实际资源尺寸。如果资源包含多个图像, 则使用第一个图像的大小 |
| LR_LOADFROMFILE     | 根据参数 lpszName 的值装载图像。若标记未被给定, lpszName 的值为资源名称                                                                                  |
| LR_LOADMAP3DCOLORS  | 查找图像的颜色表并且按相应的 3D-颜色表的灰度进行替换                                                                                                    |
| LR_LOADTRANSPARENT  | 找到图像中的一个像素颜色值并且根据颜色表中系统的默认颜色值替代其相应接口的值。图像中所有使用这种接口的像素的颜色都变为系统的默认窗体颜色。此值仅用来申请相应的颜色表                                              |
| LR_MONOCHROME       | 装载黑白图                                                                                                                           |
| LR_SHARED           | 若图像将被多次装载则共享。如果 LR_SHARED 未被设置, 则向同一个资源第二次调用这个图像时就会再装载一遍这个图像并返回不同的句柄                                                            |
| LR_VGACOLOR         | 使用 VGA 真彩色                                                                                                                      |

## 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“利用 RichEdit 控件显示图文数据”。
- (2) 向窗体中添加一个 RichEdit 控件、一个编辑框, 为 RichEdit 控件设置 Multiline、Horizontal scroll、Auto Hscroll、Vertical scroll、Want return、Border 等属性。
- (3) 在 InitInstance 函数中调用 AfxInitRichEdit 函数, 用于初始化 RichEdit 控件。
- (4) 主要程序代码如下:

```
void CNewrich::InsertBitmap(CString *pBmpFile)
{
 HBITMAP bmp;
 //创建HBITMAP
 bmp = (HBITMAP)::LoadImage(NULL, *pBmpFile, IMAGE_BITMAP, 0, 0,
 LR_LOADFROMFILE|LR_DEFAULTCOLOR|LR_DEFAULTSIZE); //加载图片资源
 //设置STGMEDIUM结构
 STGMEDIUM stgm;
 stgm.tymed = TYMED_GDI;
 stgm.hBitmap = bmp;
 stgm.pUnkForRelease = NULL;
 //设置FORMATETC结构
 FORMATETC fm;
 fm.cfFormat = CF_BITMAP;
 fm.ptd = NULL;
 fm.dwAspect = DVASPECT_CONTENT;
 fm.lindex = -1;
 fm.tymed = TYMED_GDI;
 //创建输入数据源
 IStorage *pStorage;
 //分配内存
 LPLOCKBYTES lpLockBytes = NULL;
```

```

SCODE sc = ::CreateLockBytesOnHGlobal(NULL, TRUE, &lpLockBytes);
if (sc != S_OK)
 AfxThrowOleException(sc);
ASSERT(lpLockBytes != NULL);
sc = ::StgCreateDocfileOnLockBytes(lpLockBytes,
 STGM_SHARE_EXCLUSIVE|STGM_CREATE|STGM_READWRITE, 0, &pStorage);
if (sc != S_OK)
{
 VERIFY(lpLockBytes->Release() == 0);
 lpLockBytes = NULL;
 AfxThrowOleException(sc);
}
ASSERT(pStorage != NULL);
COleDataSource *pDataSource = new COleDataSource;
pDataSource->CacheData(CF_BITMAP, &stgm);
LPDATAOBJECT lpDataObject =
 (LPDATAOBJECT)pDataSource->GetInterface(&IID_IDataObject);
//获取RichEdit的OLEClientSite
LPOLECLIENTSITE lpClientSite;
this->GetRichEditOle()->GetClientSite(&lpClientSite);
//创建OLE对象
IOleObject *pOleObject;
sc = OleCreateStaticFromData(lpDataObject, IID_IOleObject, OLERENDER_FORMAT,
 &fm, lpClientSite, pStorage, (void **)&pOleObject);
if (sc != S_OK)
 AfxThrowOleException(sc);
//插入OLE对象
REOBJECT reobject;
ZeroMemory(&reobject, sizeof(REOBJECT));
reobject.cbStruct = sizeof(REOBJECT);
CLSID clsid;
sc = pOleObject->GetUserClassID(&clsid);
if (sc != S_OK)
 AfxThrowOleException(sc);
//设置REOBJECT结构
reobject.clsid = clsid;
reobject.cp = REO_CP_SELECTION;
reobject.dvaspect = DVASPECT_CONTENT;
reobject.poleobj = pOleObject;
reobject.polesite = lpClientSite;
reobject.pstg = pStorage;
HRESULT hr = this->GetRichEditOle()->InsertObject(&reobject);
delete pDataSource;
}

```

### 举一反三

根据本实例，读者可以：

- 利用 RichEdit 控件制作画图工具。

### 实例 092

### 在 RichEdit 中显示不同 字体和颜色的文本

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\02\092

### 实例说明

RichEdit 控件和编辑框控件都可以显示文本，不同之处在于 RichEdit 控件可以显示不同的字体、颜色及图片信息。本实例就通过 RichEdit 控件显示不同字体和颜色的文本。实例运行结果如图 2.46 所示。

### 技术要点

本实例主要通过 GetDefaultCharFormat 方法、SetWord

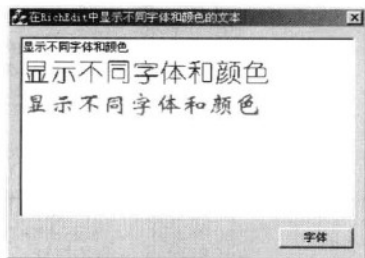


图 2.46 在 RichEdit 中显示不同字体和颜色

CharFormat 方法、SetSel 方法和 ReplaceSel 方法来实现。

(1) GetDefaultCharFormat 方法。GetDefaultCharFormat 方法用于获得 RichEdit 控件缺省的字符格式化属性, 该方法的语法格式如下:

```
DWORD GetDefaultCharFormat(CHARFORMAT& cf) const;
```

参数说明:

- cf: 指向一个 CHARFORMAT 结构的指针, 该结构将包含缺省的字符格式化属性。

(2) SetWordCharFormat 方法。SetWordCharFormat 方法用于设置 RichEdit 控件当前选择的文本的字符格式化属性, 其语法格式如下:

```
BOOL SetWordCharFormat(CHARFORMAT& cf);
```

参数说明:


- cf: 一个 CHARFORMAT 结构, 包含了当前选择的字符格式化属性。

(3) SetSel 方法。SetSel 方法用于设置 RichEdit 控件当前选择的文本, 其语法格式如下:

```
void SetSel(long nStartChar, long nEndChar);
void SetSel(CHARRANGE& cr);
```

参数说明:

- nStartChar: 标识起始位置。
- nEndChar: 标识结束位置。
- cr: 一个 CHARRANGE 结构, 包含了当前选择的界线。

 注意: 对于 SetSel 方法, 当参数为+1 和-1 时, 将选中结尾行, 当参数为 0 和-1 时将选中编辑框所有内容。

(4) ReplaceSel 方法。ReplaceSel 方法用于使用指定的文本替换 RichEdit 控件中当前选中的文本, 语法如下:

```
void ReplaceSel(LPCTSTR lpszNewText, BOOL bCanUndo = FALSE);
```

参数说明:

- lpszNewText: 用于替换的文本。
- bCanUndo: 是否具有取消操作的功能。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 向对话框中添加一个 RichEdit 控件和一个按钮控件, 设置 RichEdit 控件的 Multiline、Horizontal scroll、Auto HScroll、Vertical scroll、Auto Vscroll、Want return 属性, 添加成员变量 m\_RichEdit;

(3) 主要程序代码如下:

```
void CEditShowDlg::OnButfont()
{
 CFontDialog dlg;
 if(dlg.DoModal() != IDOK)
 {
 LOGFONT temp;
 dlg.GetCurrentFont(&temp);
 CHARFORMAT cf;
 memset(&cf, 0, sizeof(CHARFORMAT));
 m_RichEdit.GetDefaultCharFormat(cf);
 cf.yHeight = temp.lfWeight;
 cf.dwMask = CFM_COLOR | CFM_SIZE | CFM_FACE;
 cf.dwEffects = CFE_BOLD;
 cf.crTextColor = dlg.GetColor();
 strepy(cf.szFaceName, temp.lfFaceName);
 m_RichEdit.SetWordCharFormat(cf);
 m_RichEdit.SetSel(-1, -1);
 m_RichEdit.ReplaceSel("\n");
 m_RichEdit.SetSel(-1, -1);
 }
 //初始化字体信息
 //判断是否按下“确定”按钮
 //声明LOGFONT结构指针
 //获取当前字体信息
 //声明CHARFORMAT变量
 //分配内存
 //获得缺省的字符格式化属性
 //设置字号
 //设置标记属性
 //设置标记属性有效
 //设置颜色
 //设置字体
 //设置控件显示字体
 //选择最后一行
 //插入换行符
 //选择最后一行
}
```

## 举一反三

根据本实例,读者可以:

- 设计 RichEdit 控件边框和背景。

## 实例 093

## 在 RichEdit 中显示 GIF 动画

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\02\093

## 实例说明

用户在使用腾讯 OICQ 聊天软件时,就被支持各种图像格式的编辑框控件所吸引,本实例就设计了一款可以显示 GIF 动画的 RichEdit 控件。运行程序,单击“打开”按钮插入 GIF 动画,如图 2.47 所示。

## 技术要点

本实例使用 RichEdit 控件显示 GIF 动画,要实现这一功能可以分两步进行,首先设计一个 ATL 控件,然后将 ATL 控件插入到 RichEdit 控件中,下面就来介绍一下。

## 1. 设计 ATL 控件

在 Visual C++ 中设计 ATL 控件比较容易,但是显示 GIF 动画并不容易。为了降低程序的难度,本例采用了 GDI+ 实现 GIF 动画的显示。GDI+ 是微软公司 .NET 类库的一个组成部分,它并没有集成在 Visual C++ 6.0 开发环境中,但是用户可以在 Visual C++ 6.0 环境下使用 GDI+。下面介绍如何在 Visual C++ 6.0 中使用 GDI+。

(1) 下载 GDI+ 包文件。

(2) 引用 Gdiplus.h 头文件。

(3) 引用 Gdiplus 命名空间。

```
using namespace Gdiplus; //引用命名空间
```

(4) 定义两个全局变量。

```
GdiplusStartupInput m_Gdiplus;
ULONG_PTR m_pGdiToken;
```

(5) 在应用程序或对话框初始化时加载 GDI+。

```
GdiplusStartup(&m_pGdiToken,&m_Gdiplus,NULL); //初始化GDI+
```

(6) 在应用程序结束时卸载 GDI+。

```
GdiplusShutdown(m_pGdiToken); //卸载GDI+
```

(7) 在程序中链接 gdiplus.lib 库文件。

```
#pragma comment(lib,"gdiplus.lib") //链接库文件
```

(8) 显示 GIF 动画。

```
Bitmap *pBmp = Bitmap::FromFile(m_SrcFile.AllocSysString()); //根据文件名称获取图像对象
```

```
Graphics gh(di.hdcDraw);
```

```
gh.DrawImage(pBmp, rc.left+1, rc.top+1, pBmp->GetWidth(), pBmp->GetHeight()); //显示图像
```

了解了 GDI+ 的使用方法以后,就可以开始设计 ATL 控件了,设计步骤如下。

(1) 创建一个 ATC Com 工程,在向导中选择支持 MFC,如图 2.48 所示。

(2) 单击“Finish”按钮完成工程的创建。在工作区的类视图窗口中用鼠标右键单击根节点,在弹出的快捷菜单中选择“New ATL Object”菜单项,打开 ATL 对象向导窗口,如图 2.49 所示。

(3) 在 Category 列表下选择 Controls,在 Objects 列表中选择 Full Control,创建一个 ATL 控件,单击 Next 按钮,设置 ATL 控件信息,如图 2.50 所示。

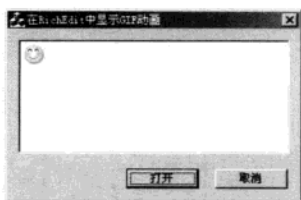


图 2.47 在 RichEdit 中显示 GIF 动画



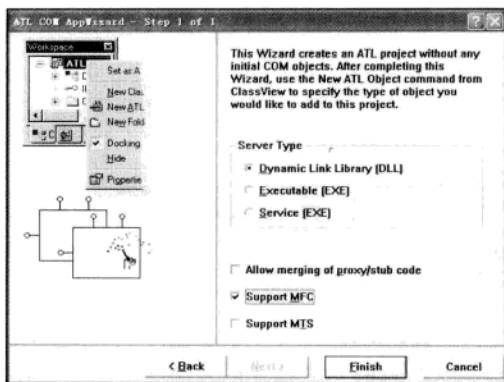


图 2.48 ATL Com 向导窗口

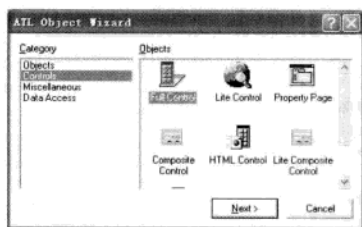


图 2.49 ATL 对象向导

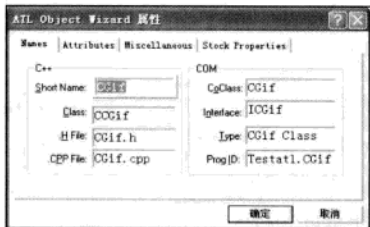


图 2.50 ATL 对象向导

- (4) 选择 Attributes 选项卡，设置 ATL 控件属性，如图 2.51 所示。
- (5) 选择 Miscellaneous 选项卡，为 ATL 控件选择一个基类，如图 2.52 所示。

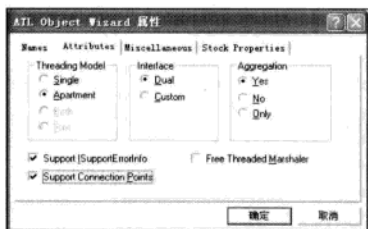


图 2.51 ATL 对象属性窗口

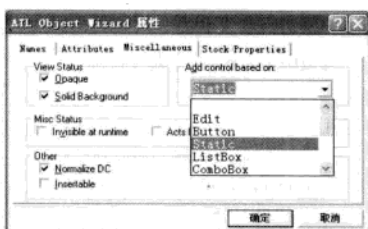


图 2.52 设置 ATL 控件基类

- (6) 单击“确定”按钮创建 ATL 控件，向 ATL 控件中添加成员变量。代码如下：

```
GdiplusStartupInput m_Gdiplus; //定义GDI+初始化变量
ULONG_PTR m_pGdiToken; //定义GDI+标识
Bitmap *m_pBmp; //定义位图对象，派生于Image类
UINT m_Count; //记录维数
UINT m_FrameCount; //帧数
PropertyItem* pItem; //定义图像属性
int fcount; //定义一个临时整型变量
UINT delay; //第一帧的延时
RECT m_RC;
static BOOL m_bChange;
static BOOL m_bNewInsert; //标记是否有对象被插入
static BOOL m_bSetHook; //标记是否挂钩
static WNDPROC RichProc; //定义窗口函数指针
int m_Num;
static int m_ImgNum; //记录插入的图像数量
static CCGif *m_ImgInfo[MAX_IMGNUM];
static HHOOK m_hHook;
static HANDLE m_hThread;
ATL_DRAWINFO m_DrawInfo;
BOOL m_bLoaded; //是否已经加载了图像
```

|             |                          |                     |
|-------------|--------------------------|---------------------|
| ImageTypeEx | m_ImageType;             |                     |
| int         | m_InsertIndex;           |                     |
| HWND        | hTmp;                    | //临时句柄, 标记ATL控件的父窗口 |
| WCHAR       | m_wchFileName[MAX_PATH]; | //记录文件名称            |
| HGLOBAL     | m_hMem;                  | //表示加载图像使用的内容空间     |
| GUID        | ImageID;                 |                     |
| BOOL        | m_bDraw;                 | //图像是否显示            |
| SYSTEMTIME  | m_DrawTime;              | //绘画时间              |

(7) 在构造函数中初始化非静态成员。代码如下:

```

m_bLoaded = FALSE;
m_bChange = FALSE;
m_ImageType = IT_UNKNOWN; //图像类型未知
m_bNewInsert = TRUE;
m_InsertIndex = 0;
hTmp = NULL;
m_hMem = NULL;
m_bDraw = FALSE;
m_Num = 0;

```

在全局区域初始化静态成员。代码如下:

```

//初始化静态成员
BOOL CCGif::m_bSetHook = FALSE;
BOOL CCGif::m_bChange = FALSE;
WNDPROC CCGif::RichProc = NULL;
int CCGif::m_ImageNum = 0;
HANDLE CCGif::m_hThread = NULL;
HHOOK CCGif::m_hHook = NULL;
CCGif *CCGif::m_ImageInfo[MAX_IMGNUM] = {0};

```

(8) 改写基类的 `IoleObject_SetClientSite` 方法, 目的是防止 `m_spClientSite` 成员为空时出错。代码如下:

```

//改写基类的IoleObject_SetClientSite方法, 设置m_bNewInsert成员
inline HRESULT IoleObject_SetClientSite(IoleClientSite *pClientSite)
{
 //防止m_spClientSite为空时弹出错误对话框
 //ATLASSERT(pClientSite == NULL || m_spClientSite == NULL);
 m_bNewInsert = TRUE;
 if (pClientSite == NULL)
 return S_OK;
 m_spClientSite = pClientSite; //根据参数设置成员变量
 m_spAmbientDispatch.Release(); //释放m_spAmbientDispatch对象
 if (m_spClientSite != NULL)
 {
 m_spClientSite->QueryInterface(IID_IDispatch,
 (void**) &m_spAmbientDispatch.p); //重新获取m_spAmbientDispatch对象信息
 }
 return S_OK;
}

```

(9) 在 ATL 控件的 `OnDraw` 方法中绘制图像, 所有的绘图操作都是在该方法中进行的。这里遇到的一个问题是对于 GIF 动画, 需要时时更新, 以显示下一帧图像。由于我们创建的 ATL 控件不是窗口控件, 不能利用 `WM_TIMER` 消息来更新图像, 这里笔者采用的方式是使用线程来更新图像。代码如下:

```

HRESULT OnDraw(ATL_DRAWINFO& di)
{
 if (m_bLoaded)
 {
 memcpy(&m_DrawInfo, &di, sizeof(ATL_DRAWINFO));
 //获取图像的显示区域
 RECT& rc = *(RECT*)&di.prcBounds;
 GUID Guid = FrameDimensionTime;
 //如果图像已经插入到控件中, 获取插入对象的父窗口句柄
 if (m_bNewInsert == TRUE)
 {
 IoleInPlaceSite* pSite = NULL;
 if (m_spClientSite != NULL)
 {
 m_spClientSite->QueryInterface(__uuidof(IoleInPlaceSite), (void**)&pSite);
 if (pSite != NULL)
 {
 pSite->GetWindow(&hTmp);
 }
 }
 }
 }
}

```



```

 m_bNewInsert = FALSE;
 }
 //是否为GIF动画
 if(m_ImageType == IT_DYNAMIC)
 {
 if (m_bSetHook == FALSE)
 {
 m_bSetHook = TRUE;
 //创建一个线程, 用于显示动画
 m_hThread = CreateThread(NULL, 0, ThreadProc, (void*)this, 0, 0);
 //设置一个消息钩子, 截获父窗口的消息, 用于更新动画
 m_hHook = ::SetWindowsHookEx(WH_GETMESSAGE, GetMsgProc, 0, ::GetCurrentThreadId());
 }
 //显示图像
 Graphics gh(di.hdcDraw);
 gh.DrawImage(m_pBmp, rc.left+1, rc.top+1, m_pBmp->GetWidth(), m_pBmp->GetHeight());
 //记录当前图像的区域
 m_RC.left = rc.left+1;
 m_RC.top = rc.top+1;
 m_RC.right = m_RC.left+m_pBmp->GetWidth();
 m_RC.bottom = m_RC.top + m_pBmp->GetHeight();
 //如果是Gif动画, 显示下一帧图像
 if (m_ImageType==IT_DYNAMIC)
 {
 if (m_Num == 0)
 {
 m_pBmp->SetActiveFrame(&Guid, fcount++);
 }
 if(fcount >= m_FrameCount)
 {
 fcount = 0;
 }
 }
 }
 return S_OK;
}

```

(10) 编写线程函数, 向 ATL 控件的父窗口发送更新窗口的消息, 实现 GIF 动画的显示。  
代码如下:

```

static DWORD __stdcall ThreadProc(LPVOID lpParameter)
{
 while (true)
 {
 Sleep(250); //演示250毫秒
 for (int i=0; i< m_ImageNum; i++) //遍历已经插入的ATL控件
 {
 if (m_ImageInfo[i] != NULL)
 {
 if (m_ImageInfo[i]->m_ImageType == IT_DYNAMIC) //如果是GIF动画
 {
 //更新父窗口的局部区域
 ::InvalidateRect(m_ImageInfo[i]->hTmp, &m_ImageInfo[i]->m_RC, FALSE);
 m_ImageInfo[i]->m_Num = 0;
 }
 }
 }
 m_bChange = FALSE;
 }
 return 0;
}

```

(11) 编写钩子函数, 当用户在多功能编辑控件中移动鼠标或者按下按键时刷新窗口, 目的是显示下一帧 GIF 动画。在向多功能编辑框中插入 GIF 动画时, 如果用户在编辑框中输入文本或者选中文本, GIF 图像会快速的切换, 为此, 笔者设计了一个钩子, 截获多功能编辑框的按键消息。代码如下:

```

static LRESULT __stdcall GetMsgProc(int code, WPARAM wParam, LPARAM lParam)
{
 MSG *pMsg = (MSG*)lParam;
 if (pMsg != NULL)
 {

```

```
char cName[MAX_PATH] = {0};
GetClassName(pMsg->hwnd, cName, MAX_PATH);
if (pMsg->message == WM_KEYDOWN || pMsg->message == WM_MOUSEMOVE)
{
 for (int i=0; i< m_ImageNum; i++)
 {
 if (m_ImageInfo[i] != NULL)
 {
 if (m_ImageInfo[i]->hTmp == pMsg->hwnd)
 {
 m_ImageInfo[i]->m_Num = 1;
 break;
 }
 }
 }
 return S_OK;
}
else if (pMsg->message == WM_PAINT)
{
 for (int i=0; i< m_ImageNum; i++)
 {
 if (m_ImageInfo[i] != NULL && m_ImageInfo[i]->m_Num == 1)
 {
 if (m_ImageInfo[i]->hTmp == pMsg->hwnd)
 {
 m_ImageInfo[i]->m_Num = 2;
 break;
 }
 }
 }
 return S_OK;
}
}
return CallNextHookEx(0, code, wParam, lParam);
}
```

(12) 向 ATL 控件的接口中添加 LoadFromFile 方法, 加载并显示图像文件。当客户端向多功能编辑控件中插入 ATL 控件后, 需要调用该方法来显示图像。代码如下:

STDMETHODIMP CCGif::LoadFromFile(LPCTSTR FileName)

```
{
 AFX_MANAGE_STATE(AfxGetStaticModuleState())
 m_bChange = FALSE;
 RECT rc;
 rc.left = 2; //定义区域对象
 rc.top = 2; //初始化区域对象的大小
 rc.right = 30;
 rc.bottom = 30;
 GdiplusStartup(&m_pGdiToken, &m_Gdiplus, NULL); //初始化GDI+
 m_FileName = (BSTR)FileName; //记录文件名
 if (m_hMem != NULL) //判断之前是否分配了内存
 {
 GlobalFree(m_hMem); //释放内存空间
 m_hMem = NULL; //初始化成员m_hMem
 }
 //获取宽字节字符串转换为多字节字符串的长度
 int nLen = WideCharToMultiByte(CP_ACP, 0, (unsigned short*)FileName, -1, NULL, 0, NULL, NULL);
 char* pData = new char[nLen]; //定义字符缓冲区
 //将宽字节字符串转换为多字节字符串
 WideCharToMultiByte(CP_ACP, 0, (unsigned short*)FileName, -1, pData, nLen, NULL, NULL);
 CFile file; //定义文件对象
 file.Open(pData, CFile::modeRead); //以读方式打开文件
 DWORD dwLen = file.GetLength(); //获取文件长度
 delete []pData; //释放字符缓冲区
 m_hMem = GlobalAlloc(GMEM_FIXED, dwLen); //为文件数据分配内存
 BYTE * pData = (BYTE*)GlobalLock(m_hMem); //获取内存空间的指针
 file.ReadHuge(pData, dwLen); //将文件数据读入内存
 file.Close(); //关闭文件
 IStream* pStm = NULL; //定义流接口指针
 CreateStreamOnHGlobal(m_hMem, FALSE, &pStm); //在堆中创建流对象
 m_pBmp = Bitmap::FromStream(pStm); //从流中获取位图对象
 GlobalUnlock(m_hMem); //解锁内存空间
 if (m_pBmp != NULL) //判断位图对象是否为空
 {
 m_pBmp->GetRawFormat(&ImageID); //获取位图的GUID, 判断图像类型
 if (ImageID == ImageFormatGIF) //是否为GIF图像
 }
}
```





```

 {
 m_ImageType = IT_DYNAMIC; //设置图像类型
 }
 else if (ImageID==ImageFormatBMP || ImageID==ImageFormatIcon ||
 ImageID==ImageFormatJPEG) //是否为静态图像
 {
 m_ImageType = IT_STATIC; //设置图像类型
 }
 if(m_ImageType==IT_DYNAMIC) //如果是GIF图像
 {
 //获取GIF图像维数
 m_Count = m_pBmp->GetFrameDimensionsCount();
 GUID *pGuids = new GUID[m_Count]; //定义一个GUID数组
 //获取帧列表
 m_pBmp->GetFrameDimensionsList(pGuids,m_Count);
 //获取图像帧数
 m_FrameCount = m_pBmp->GetFrameCount(pGuids);
 UINT size = 0;
 delay = PropertyTagFrameDelay;
 m_Count = 0;
 //获取图像属性
 m_pBmp->GetPropertySize(&size,&delay);
 PropertyItem *pItem = NULL; //定义属性对象
 pItem = (PropertyItem*)malloc(size); //为属性对象分配空间
 //获取所有属性
 Status status = m_pBmp->GetAllPropertyItems(size,delay,pItem);
 delay = ((long*)pItem->value)[1]; //获取第一帧的延时
 free(pItem); //释放属性对象
 delete [] pGuids; //释放GUID列表
 fcount = 0;
 }
 if (m_ImageType==IT_UNKNOWN) //图像格式不正确
 return S_OK; //不加载图像m_bLoaded为FALSE
 m_rcPos.left = 1; //设置图像的显示区域
 m_rcPos.top = 1;
 m_rcPos.right = m_pBmp->GetWidth()+1;
 m_rcPos.bottom = m_pBmp->GetHeight()+1;
 CSize pixelsize,newsize;
 pixelsize.cx = m_pBmp->GetWidth()+2;
 pixelsize.cy = m_pBmp->GetHeight()+2;

 AtlPixelToHiMetric(&pixelsize,&newsize);
 m_sizeExtent.cx = newsize.cx;
 m_sizeExtent.cy = newsize.cy;
 SendOnViewChange(DVASPECT_CONTENT); //重新修改显示区域, m_sizeExtent成员生效
 m_bLoaded = TRUE;
 m_bNewInsert = TRUE;
 m_ImageNum++;
 m_ImageInfo[m_ImageNum-1] = this;
 hTmp = NULL;
}
return S_OK;
}

```

(13) 向 ATL 控件接口中添加 SaveToFile 方法, 用于保存图像到磁盘文件中。因为在客户端当用户接收或发送图像时, 可以在编辑框中用鼠标右键单击图像, 在弹出的快捷菜单中有一个“收藏图片”菜单, 单击该菜单将调用 SaveToFile 方法保存图像到文件中。代码如下:

STDMETHODIMP CCGif::SaveToFile(BSTR FileName)

```

{
 AFX_MANAGE_STATE(AfxGetStaticModuleState())
 CLSID clsid;
 int nRet = -1;
 if (ImageID == ImageFormatGIF) //是否为GIF图像
 {
 nRet = GetCodecClsid(L"image/gif", &clsid); //获取GUID
 }
 else if (ImageID==ImageFormatBMP) //是否为位图图像
 {
 nRet = GetCodecClsid(L"image/bmp", &clsid); //获取GUID
 }
 else if (ImageID==ImageFormatIcon) //是否为图标
 {
 nRet = GetCodecClsid(L"image/bmp", &clsid); //获取GUID
 }
}

```

```

else if (ImageID==ImageFormatJPEG) //是否为JPEG
{
 nRet = GetCodecClsid(L"image/jpeg", &clsid); //获取GUID
}
if (nRet== -1)
 return S_FALSE;

IStream *pstm=NULL; //定义流接口指针
CreateStreamOnHGlobal(m_hMem, TRUE, &pstm); //在堆中创建流
//获取宽字节转换为多字节字符串的长度
int nLen = WideCharToMultiByte(CP_ACP,0,(unsigned short*)FileName,-1,NULL,0,NULL,NULL);
char* pdata = new char[nLen]; //定义字符串缓冲区
//将宽字节字符串转换为多字节字符串
WideCharToMultiByte(CP_ACP,0,(unsigned short*)FileName,-1,pdata,nLen,NULL,NULL);
CFile file; //定义文件对象
file.Open(pdata,CFile::modeCreate|CFile::modeReadWrite); //创建文件
BYTE* pData = (BYTE*)GlobalLock(m_hMem); //获取图像数据
DWORD dwSize = GlobalSize(m_hMem); //获取图像数据的大小
file.WriteHuge(pData,dwSize); //写入图像数据到文件中
file.Close(); //关闭文件
delete []pdata; //释放字符串缓冲区
GlobalUnlock(m_hMem); //解锁堆空间
return S_OK;
}

```

(14) 向 ATL 控件接口中添加 FileName 属性。其实现代码如下：

```

STDMETHODIMP CCGif::get_FileName(BSTR *pVal)
{
 AFX_MANAGE_STATE(AfxGetStaticModuleState())
 *pVal = m_FileName //获取图像的文件名称
 return S_OK;
}

STDMETHODIMP CCGif::put_FileName(BSTR newVal)
{
 AFX_MANAGE_STATE(AfxGetStaticModuleState())
 m_FileName = newVal; //设置图像的文件名称
 LoadFromFile(LPCTSTR)m_FileName; //加载图像
 return S_OK;
}

```

至此，就完成了 ATL 控件的设计。

## 2. 将 ATL 控件插入到 RichEdit 控件中

CRichEditCtrl 控件插入 ATL 控件的主要思路是通过 IRichEditOle 接口的 InsertObject 方法实现的。用户可以使用 CRichEditCtrl 控件的 GetIRichEditOle 方法获取 IRichEditOle 接口指针。所有的插入操作都是围绕 InsertObject 方法的参数进行的。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加一个 RichEdit 控件，并为其添加成员变量 m\_RichEdit。
- (3) 主要程序代码如下：

```

void CTestGifDlg::OnOK()
{
 CFileDialog fDlg(TRUE, "", "", OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
 "图片文件*.bmp;*.gif;*.jpg;*.jpeg;*.ico;|", this); //构造打开对话框
 if (fDlg.DoModal()==IDOK)
 {
 CString csFile = fDlg.GetPathName(); //获得GIF动画路径
 IRichEditOle *lpRichOle = m_RichEdit.GetIRichEditOle(); //获得IRichEditOle接口指针
 if (lpRichOle != NULL)
 {
 InsertImage(lpRichOle,csFile); //插入ATL控件
 lpRichOle->Release();
 lpRichOle = NULL;
 }
 }
}

BOOL CTestGifDlg::InsertImage(IRichEditOle *lpRichEditOle, CString &csFileName)
{

```

```

IStorage *lpStorage = NULL; //存储接口
IOleObject *lpOleObject = NULL; //定义Ole对象指针
LPLOCKBYTES lpLockBytes = NULL; //定义LOCKBYTES指针,用于创建存储对象
IOleClientSite *lpOleClientSite = NULL; //定义IOleClientSite接口指针
GIFLib::ICGifPtr lpAnimator; //定义ATL控件接口指针
CLSID clsid; //定义类ID对象
REOBJECT reobject; //定义InsertObject方法的参数
HRESULT hr;
if (lpRichEditOle == NULL)
{
 return FALSE;
}
hr = ::CoInitialize(NULL); //初始化Com
if (FAILED(hr))
{
 _com_issue_error(hr);
}
hr = lpAnimator.CreateInstance(GIFLib::CLSID_CGif); //创建ATL控件实例
if (FAILED(hr))
{
 _com_issue_error(hr);
}
lpRichEditOle->GetClientSite(&lpOleClientSite); //获取IOleClientSite
try
{
 //获取OLE对象接口
 hr = lpAnimator->QueryInterface(IID_IOleObject, (void**)&lpOleObject);
 if (FAILED(hr))
 {
 AfxMessageBox("Error QueryInterface");
 }
 hr = lpOleObject->GetUserClassID(&clsid); //获取类ID
 if (FAILED(hr))
 {
 AfxMessageBox("Error GetUserClassID");
 }
 lpOleObject->SetClientSite(NULL); //防止出现错误提示
 lpOleObject->SetClientSite(lpOleClientSite); //设置ATL控件的OleClientSite
 hr = ::CreateLockBytesOnHGlobal(NULL, TRUE, &lpLockBytes); //创建LOCKBYTES对象
 if (FAILED(hr))
 {
 AfxThrowOleException(hr);
 }
 ASSERT(lpLockBytes != NULL);
 hr = ::StgCreateDocfileOnILockBytes(lpLockBytes, STGM_SHARE_EXCLUSIVE | STGM_CREATE |
 STGM_READWRITE, 0, &lpStorage); //创建根存储对象
 if (FAILED(hr))
 {
 VERIFY(lpLockBytes->Release() == 0);
 lpLockBytes = NULL;
 AfxThrowOleException(hr);
 }
 ZeroMemory(&reobject, sizeof(REOBJECT)); //初始化参数对象
 reobject.cbStruct = sizeof(REOBJECT); //设置结构的大小
 reobject.clsid = clsid; //设置类ID
 reobject.cp = REO_CP_SELECTION;
 reobject.dvaspect = DVASPECT_CONTENT;
 reobject.dwFlags = REO_BLANK;
 reobject.poleobj = lpOleObject; //设置Ole对象
 reobject.polesite = lpOleClientSite; //设置OleClientSite
 reobject.pstg = lpStorage; //设置根存储
 hr = lpRichEditOle->InsertObject(&reobject); //插入对象
 hr = lpAnimator->LoadFromFile(csFileName, AllocSysString()); //加载文件
 if (FAILED(hr))
 {
 AfxThrowOleException(hr);
 }
 RedrawWindow(); //刷新窗体
 lpOleClientSite->SaveObject(); //保存Ole对象
 OleSetContainedObject(lpOleObject, TRUE); //设置容器对象
}
catch (CException* e)
{
 e->Delete();
}

```



```
lpAnimator->Release();
lpStorage->Release();
return TRUE;
```

```
//释放ATL接口指针
//释放存储接口指针
```

## 举一反三

根据本实例,读者可以:

- 利用 RichEdit 控件制作画图工具。

## 2.9 滚动条控件典型实例

## 实例 094 自定义滚动条控件

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\094

## 实例说明

网上的许多应用软件都具有漂亮的滚动条,例如,著名的腾讯 OICQ 软件。本实例利用 CStatic 控件设计一个自定义的滚动条,效果如图 2.53 所示。

## 技术要点

要实现自定义滚动条控件,主要有 3 种方法。一是利用钩子技术重新绘制滚动条,该方法实现起来比较复杂。二是获得滚动条的显示区域,将其扣除,然后在该区域显示自定义的滚动条控件。三是自定义一个滚动条控件,将其与对话框中的某个控件关联,在创建滚动条控件时,将对话框中的某个控件隐藏,并在该控件的位置显示滚动条控件。本实例采用第 3 种方法。利用 CStatic 控件派生一个自定义滚动条 CCustomScroll,在 CStatic 控件上利用位图绘制滚动条箭头、滚动块及滚动条的滚动区域。在绘制滚动条时,由于滚动块能够被拖动,因此需要频繁地绘制滚动条。为了防止出现屏幕的闪烁,可以定义一个临时的 CDC 对象,将所有的绘图操作都在该临时对象上进行,然后再将临时对象的内容绘制在滚动块的显示区域。为了简化操作,本实例将临时的 CDC 对象的功能封装为 CMemDC 类,在该类释放时会自动将其自身的内容绘制到某一个显示区域上。

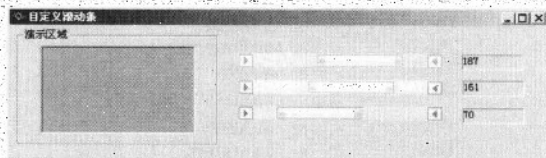


图 2.53 自定义滚动条控件

```
class CMemDC : public CDC
{
private:
 CBitmap* m_bitmap;
 CBitmap* m_oldbmp;
 CDC* m_pDC;
 CRect m_Rect;
public:
 CMemDC(CDC* pDC, const CRect& rect) : CDC()
 {
 CreateCompatibleDC(pDC);
 m_bitmap = new CBitmap;
 m_bitmap->CreateCompatibleBitmap(pDC, rect.Width(), rect.Height());
 m_oldbmp = SelectObject(m_bitmap);
 m_pDC = pDC;
 m_Rect = rect;
 }
 ~CMemDC()
 {
 m_pDC->BitBlt(m_Rect.left, m_Rect.top, m_Rect.Width(),
 m_Rect.Height(),
 this, m_Rect.left, m_Rect.top, SRCCOPY);
 }
};
```



```
SelectObject(m_oldbmp);
if (m_bmp != NULL)
 delete m_bmp;
};
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加按钮、编辑框、静态文本控件。
- (3) 向对话框类中添加成员变量：

|        |                 |                |
|--------|-----------------|----------------|
| UINT   | m_ThumbWidth;   | //滚动块和箭头宽度     |
| UINT   | m_ThumbHeight;  | //滚动块和箭头高度     |
| CWnd*  | m_pParent;      | //父窗口          |
| CRect  | m_ClientRect;   | //窗口客户区域       |
| CRect  | m_ThumbRect;    | //滚动块区域        |
| BOOL   | m_ButtonDown;   | //鼠标是否单击滚动块    |
| CPoint | m_Startpt;      | //鼠标按下时的起点     |
| BOOL   | m_IsLeft;       | //滚动块是否超过左箭头   |
| BOOL   | m_IsLeftArrow;  | //是否单击左滚动条按钮   |
| BOOL   | m_IsRightArrow; | //是否单击右滚动条按钮   |
| BOOL   | m_IsLeftRange;  | //是否单击了左滚动区域   |
| BOOL   | m_IsRightRange; | //是否单击了右滚动区域   |
| UINT   | m_MinRange;     | //最小滚动范围       |
| UINT   | m_MaxRange;     | //最大滚动范围       |
| UINT   | m_CurPos;       | //当前的位置(逻辑单位)  |
| double | m_Rate;         | //物理像素与逻辑单位的比率 |
| UINT   | m_LeftArrow;    | //左箭头位图ID      |
| UINT   | m_RightArrow;   | //右箭头位图ID      |
| UINT   | m_ChanelBK;     | //背景位图ID       |
| UINT   | m_ThumbBK;      | //滚动块位图ID      |

- (4) 在 CCustomScroll 类的构造函数中初始化成员变量，代码如下：

```
CCustomScroll::CCustomScroll()
{
 m_ButtonDown = FALSE;
 m_IsLeft = FALSE;
 m_MinRange = 0;
 m_MaxRange = 200;
 m_CurPos = 0;
 m_IsLeftArrow = FALSE;
 m_IsRightArrow = FALSE;
 m_IsLeftRange = FALSE;
 m_IsRightRange = FALSE;
}
```

- (5) 向 CCustomScroll 类中添加 CreateStatic 成员函数，用于创建滚动条控件，代码如下：

```
BOOL CCustomScroll::CreateStatic(CWnd *pParent, DWORD dwStyle, UINT nIDStatic,
UINT nID)
{
 m_pParent = pParent;
 ASSERT(m_pParent);
 //获取父窗口中的静态文本
 ASSERT(::IsWindow(pParent->GetDlgItem(nIDStatic)->m_hWnd));
 CRect recttemp;
 pParent->GetDlgItem(nIDStatic)->GetWindowRect(recttemp); //获得窗口区域
 pParent->ScreenToClient(&recttemp); //转换为客户坐标
 m_ClientRect = recttemp;
 pParent->GetDlgItem(nIDStatic)->ShowWindow(SW_HIDE); //隐藏窗口
 //判断滚动条类型
 m_IsHor = (dwStyle&SBS_VERT)? FALSE: TRUE;
 BOOL ret = CStatic::Create("",dwStyle,m_ClientRect,pParent,nID);
 pParent->GetDlgItem(nIDStatic)->GetClientRect(m_ClientRect);

 if (ret)
 {
 CBitmap bmp;
 bmp.LoadBitmap(m_LeftArrow); //加载图片资源
 BITMAP blnfo;
```

```

bmp.GetBitmap(&bInfo); //获得图片
m_ThumbHeight = bInfo.bmHeight; //图片高度
m_ThumbWidth = bInfo.bmWidth; //图片宽度
if (bmp.GetSafeHandle())
 bmp.DeleteObject();

bmp.LoadBitmap(IDB_THUMB);
bmp.GetBitmap(&bInfo);

m_ThumbRect.CopyRect(CRect(m_ThumbWidth,0,m_ThumbWidth
+bInfo.bmWidth,bInfo.bmHeight));
if (bmp.GetSafeHandle())
 bmp.DeleteObject();

SetScrollRange(m_MinRange,m_MaxRange); //设置滚动条范围
}
ShowWindow(SW_SHOW); //显示控件
return ret;
}

```

(6) 向 CCustomScroll 类中添加 DrawHorScroll 方法, 绘制滚动条, 代码如下:

```

void CCustomScroll::DrawHorScroll()
{
 CClientDC dc(this);
 CMemDC memdc(&dc,m_ClientRect);
 CDC bmpdc;
 bmpdc.CreateCompatibleDC(&dc);
 CBitmap bmp;
 bmp.LoadBitmap(m_LeftArrow); //加载图片资源
 CBitmap* pOldbmp = bmpdc.SelectObject(&bmp); //选入位图对象
 CRect LeftArrowRect (m_ClientRect.left,m_ClientRect.top,m_ClientRect.left+
 m_ThumbWidth,m_ClientRect.bottom); //设置绘制区域
 memdc.StretchBlt(m_ClientRect.left,m_ClientRect.top,m_ThumbWidth,
 m_ThumbHeight,&bmpdc,0,0,m_ThumbWidth,m_ThumbHeight,SRCCOPY); //绘制图片
 if (pOldbmp)
 bmpdc.SelectObject(pOldbmp);
 if (bmp.GetSafeHandle())
 bmp.DeleteObject();
 pOldbmp = NULL;
 //通道的开始位置和宽度
 int nChanelStart = m_ClientRect.left+m_ThumbWidth;
 int nChanelWidth = m_ClientRect.Width()- 2*m_ThumbWidth;
 //绘制通道
 bmp.LoadBitmap(m_ChanelBK);
 pOldbmp = bmpdc.SelectObject(&bmp);
 memdc.StretchBlt(nChanelStart,m_ClientRect.top,nChanelWidth,
 m_ClientRect.Height(),&bmpdc,0,0,1,10,SRCCOPY);
 if (pOldbmp)
 bmpdc.SelectObject(pOldbmp);
 if (bmp.GetSafeHandle())
 bmp.DeleteObject();

 //绘制右箭头
 bmp.LoadBitmap(m_RightArrow);
 pOldbmp = bmpdc.SelectObject(&bmp);
 int nRArrowStart = m_ThumbWidth+nChanelWidth;
 memdc.StretchBlt(nRArrowStart,m_ClientRect.top,m_ThumbWidth,
 m_ClientRect.Height(),&bmpdc,0,0,m_ThumbWidth,m_ThumbHeight,SRCCOPY);
 //绘制滚动块
 if (bmp.GetSafeHandle())
 bmp.DeleteObject();
 bmp.LoadBitmap(m_ThumbBK);
 pOldbmp = bmpdc.SelectObject(&bmp);
 memdc.StretchBlt(m_ThumbRect.left,m_ThumbRect.top,
 m_ThumbRect.Width()+1,m_ThumbRect.Height(),&bmpdc,0,0,
 m_ThumbRect.Width(),m_ThumbRect.Height(),SRCCOPY);
}

```

(7) 处理 CCustomScroll 类的 WM\_LBUTTONDOWN 消息, 根据用户单击的不同区域移动滚动块, 代码如下:

```

void CCustomScroll::OnLButtonDown(UINT nFlags, CPoint point)
{
 m_Startpt = point;
 //确定滚动区域
}

```



```

CRect rcScroll = m_ClientRect;
rcScroll.left += m_ThumbWidth;
rcScroll.right -= m_ThumbWidth;
DWORD wparam;
SetCapture(); //捕获鼠标
if (m_ThumbRect.PtInRect(point))
{
 m_ButtonDown = TRUE;
}
else if (rcScroll.PtInRect(point)) //单击滚动区域
{
 CPoint centerPt = m_ThumbRect.CenterPoint();
 int offset = point.x - centerPt.x;
 if ((int)point.x < m_ThumbRect.left) //左滚动区域
 m_IsLeftRange = TRUE;
 if ((int)point.x > m_ThumbRect.right)
 m_IsRightRange = TRUE;
 m_ThumbRect.OffsetRect(offset, 0);
 int left = m_ThumbRect.left;
 int right = m_ThumbRect.right;
 if (left < (int)m_ThumbWidth) //判断当前滚动量是否超出了滚动范围
 {
 int width = m_ThumbRect.Width();
 m_ThumbRect.left = m_ThumbWidth;
 m_ThumbRect.right = m_ThumbRect.left + width;
 m_CurPos = m_MinRange;
 wparam = MAKELONG(SB_PAGELEFT, m_CurPos);
 ::SendMessage(GetParent()->m_hWnd, WM_HSCROLL, wparam, (LPARAM)m_hWnd);
 DrawControl();
 return;
 }
 else if (right > (int)(m_ClientRect.Width() - m_ThumbWidth))
 {
 int width = m_ThumbRect.Width();
 m_ThumbRect.right = m_ClientRect.Width() - m_ThumbWidth;
 m_ThumbRect.left = m_ThumbRect.right - width;
 m_CurPos = m_MaxRange;
 wparam = MAKELONG(SB_PAGERIGHT, m_CurPos);
 ::SendMessage(GetParent()->m_hWnd, WM_HSCROLL, wparam, (LPARAM)m_hWnd);
 DrawControl();
 return;
 }
 int range = m_ThumbRect.left - m_ThumbWidth;
 m_CurPos = m_Rate * (range);
 if (m_IsLeftRange)
 {
 wparam = MAKELONG(SB_PAGELEFT, m_CurPos);
 ::SendMessage(GetParent()->m_hWnd, WM_HSCROLL, wparam, (LPARAM)m_hWnd);
 }
 else if (m_IsRightRange)
 {
 wparam = MAKELONG(SB_PAGERIGHT, m_CurPos);
 ::SendMessage(GetParent()->m_hWnd, WM_HSCROLL, wparam, (LPARAM)m_hWnd);
 }
 DrawControl();
}
else //单击箭头按钮
{
 if (point.x <= (int)m_ThumbWidth) //单击左箭头
 {
 if (m_CurPos > m_MinRange)
 wparam = MAKELONG(SB_LINELEFT, 1);
 else
 wparam = MAKELONG(SB_LINELEFT, 0);
 ::SendMessage(GetParent()->m_hWnd, WM_HSCROLL, wparam, (LPARAM)m_hWnd);
 if (m_CurPos > m_MinRange)
 m_CurPos--;
 m_IsLeftArrow = TRUE;
 }
 else //单击右箭头
 {
 if (m_CurPos <= m_MaxRange)
 wparam = MAKELONG(SB_LINERIGHT, 0);
 else
 wparam = MAKELONG(SB_LINERIGHT, 1);
 ::SendMessage(GetParent()->m_hWnd, WM_HSCROLL, wparam, (LPARAM)m_hWnd);
 if (m_CurPos < m_MaxRange)

```

```

 m_CurPos+=1;
 m_IsRightArrow = TRUE;
 }
 int factpos = m_CurPos/m_Rate;
 int width = m_ThumbRect.Width();
 m_ThumbRect.left = m_ThumbWidth + factpos;
 m_ThumbRect.right = m_ThumbRect.left+width;
 DrawControl();
 SetTimer(1,100,NULL); //设置定时器
}
CStatic::OnLButtonDown(nFlags, point);

```

(8) 处理 CCustomScroll 类的 WM\_MOUSEMOVE 消息, 如果用户正在拖动滚动块, 将滚动块移动到适当的位置, 代码如下:-

```

void CCustomScroll::OnMouseMove(UINT nFlags, CPoint point)
{
 if (m_ButtonDown)
 {
 int offset = point.x-m_Startpt.x;
 m_Startpt = point;
 DWORD wparam;
 if (offset<=0) //向左拖动滚动块
 {
 if (m_ThumbRect.left<=(int)m_ThumbWidth)
 return;
 else if (abs(offset)>(int)(m_ThumbRect.left-m_ThumbWidth)) //判断当前滚动条是否超出了滚动范围
 {
 int width = m_ThumbRect.Width();
 m_ThumbRect.left = m_ThumbWidth;
 m_ThumbRect.right = m_ThumbRect.left+width;
 m_CurPos = 0;
 wparam = MAKELONG(SB_THUMBPOSITION,m_CurPos);
 ::SendMessage(GetParent()->m_hWnd,WM_HSCROLL, wparam,(LPARAM)m_hWnd);
 DrawControl();
 return;
 }
 }
 else if (offset>0) //向右拖动滚动块
 {
 if (m_ThumbRect.right>=m_ClientRect.Width()-m_ThumbWidth) //超出右箭头
 {
 return;
 }
 else if (offset> m_ClientRect.Width()-m_ThumbWidth-m_ThumbRect.right) //判断当前滚动条是否超出了滚动范围
 {
 int width = m_ThumbRect.Width();
 m_ThumbRect.right = m_ClientRect.Width()-m_ThumbWidth;
 m_ThumbRect.left = m_ThumbRect.right -width;
 m_CurPos = m_MaxRange;
 wparam = MAKELONG(SB_THUMBPOSITION,m_CurPos);
 ::SendMessage(GetParent()->m_hWnd,WM_HSCROLL, wparam,(LPARAM)m_hWnd);
 DrawControl();
 return;
 }
 }
 m_ThumbRect.OffsetRect(offset,0);
 int range = m_ThumbRect.left-m_ThumbWidth;
 m_CurPos = m_Rate*(range);
 wparam = MAKELONG(SB_THUMBPOSITION,m_CurPos);
 ::SendMessage(GetParent()->m_hWnd,WM_HSCROLL, wparam,(LPARAM)m_hWnd);
 DrawHorScroll();
 }
 CStatic::OnMouseMove(nFlags, point);
}

```

(9) 处理对话框的 WM\_TIMER 消息, 当用户按下滚动条两端的箭头时, 连续地移动滚动块, 代码如下:

```

void CCustomScroll::OnTimer(UINT nIDEvent)
{
 DWORD wparam;
 if (m_IsLeftArrow) //向左移动
 {
 if (m_CurPos>m_MinRange)
 wparam = MAKELONG(SB_LINELEFT,1);
 else
 wparam = MAKELONG(SB_LINELEFT,0);
 }
}

```





```

::SendMessage(GetParent()->m_hWnd, WM_HSCROLL, wparam, (LPARAM)m_hWnd);
if (m_CurPos>m_MinRange)
 m_CurPos-=1;
}
else if (m_IsRightArrow) //向右移动
{
 if (m_CurPos<m_MaxRange)
 wparam = MAKELONG(SB_LINERIGHT,1);
 else
 wparam = MAKELONG(SB_LINERIGHT,0);
 ::SendMessage(GetParent()->m_hWnd, WM_HSCROLL, wparam, (LPARAM)m_hWnd);
 if (m_CurPos<m_MaxRange)
 m_CurPos+=1;
}
int factpos = m_CurPos/m_Rate;
int width = m_ThumbRect.Width();
m_ThumbRect.left = m_ThumbWidth + factpos;
m_ThumbRect.right = m_ThumbRect.left+width;
DrawControl();

CStatic::OnTimer(nIDEvent);
}

```

### 举一反三

根据本实例，读者可以：

- 自定义树视图控件。

## 2.10 进度条控件典型实例

进度条控件（Progress）用于显示程序的进度，在进行程序安装、文件传输时经常用到。

### 实例 095 进度条百分比显示

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\02\095

#### 实例说明

在设计应用程序时，通常使用进度条来描述当前的操作进度。但是 MFC 提供的进度条控件不能够利用精确的数字或百分比来描述进度。这要怎么解决呢？本实例实现了进度条百分比显示的功能，运行结果如图 2.54 所示。

#### 技术要点

在进度条控件中显示文字比较简单，只需要在进度条控件的 OnPaint 方法中根据当前的位置值输出字符串文本就可以了。为了提高进度条窗口的绘制效率，这里使用了 BeginPaint 方法来获得进度条窗口的设备上下文，在进度条窗口的设备上下文使用后，调用 EndPaint 方法来结束进度条窗口的绘制。下面介绍这两个方法的使用。

(1) BeginPaint 方法。BeginPaint 方法用于为窗口准备绘制操作，将绘制的信息填充到参数中。语法如下：

```
HDC BeginPaint(LPPAINTSTRUCT lpPaint);
```

参数说明：

- lpPaint：是一个 PAINTSTRUCT 结构指针，表示接收的绘制信息。

返回值：表示关联窗口的设备上下文指针。

(2) EndPaint 方法。EndPaint 方法用于表示窗口的绘制操作结束。语法如下：

```
void EndPaint(LPPAINTSTRUCT lpPaint);
```



图2.54 进度条百分比显示

参数说明:

- lpPaint: 是一个 PAINTSTRUCT 结构指针, 包含了由 BeginPaint 方法获取的绘制信息。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 从 CProgressCtrl 类派生一个子类 CTextProgress。
- (3) 向对话框中添加一个进度条控件, 设置控件的 Border 和 Smooth 属性, 关联变量 m\_Progress, 其类型为 CTextProgress。

(4) 向 CTextProgress 类中添加成员变量。代码如下:

```
COLORREF m_crText; //文本颜色
COLORREF m_crProgress; //进度颜色
COLORREF m_crBlank; //空白区域颜色
```

(5) 处理进度条的 WM\_PAINT 消息, 在其消息处理函数中绘制进度条的文本和当前进度。

代码如下:

```
void CTextProgress::OnPaint()
{
 PAINTSTRUCT ps;
 CDC* pDC = BeginPaint(&ps); //开始绘制
 int nPos = GetPos(); //获取当前进度条的位置
 CString csPos;
 csPos.Format("%d%%", nPos); //格式化字符串
 CRect clientRC;
 GetClientRect(clientRC); //获取客户区域
 CSize szText = pDC->GetTextExtent(csPos); //获取字符串的高度和宽度
 int nX = (clientRC.Width() - szText.cx) / 2; //计算中心位置
 int nY = (clientRC.Height() - szText.cy) / 2;
 pDC->SetBkMode(TRANSPARENT); //将设备上下文的背景模式设置为透明
 int nMin, nMax;
 GetRange(nMin, nMax); //获取进度条的显示范围
 //获取单位刻度
 double dFraction = (double)clientRC.Width() / (nMax - nMin);
 int nLeft = nPos * dFraction; //计算左边距
 CRect leftRC = clientRC;
 leftRC.right = nLeft;
 CRect rightRC = clientRC;
 rightRC.left = nLeft;
 pDC->FillRect(leftRC, &CBrush(m_crProgress)); //使用蓝色标识当前的进度
 pDC->FillRect(rightRC, &CBrush(m_crBlank)); //使用白色标识剩余的部分
 pDC->SetTextColor(m_crText); //设置文本颜色
 pDC->TextOut(nX, nY, csPos); //输出当前的进度
 ReleaseDC(pDC); //释放设备上下文
 EndPaint(&ps); //结束窗口绘制
}
```

(6) 处理主窗口的 WM\_TIMER 消息, 在消息处理函数中设置进度条的显示进度。代码如下:

```
void CTextProgressDlg::OnTimer(UINT nIDEvent)
{
 int nCurPos = m_Progress.GetPos(); //获取进度条的当前位置
 m_Progress.SetPos(nCurPos+1); //设置进度条的位置
 CDialog::OnTimer(nIDEvent);
}
```

## 举一反三

根据本实例, 读者可以:

- 在滑块显示的进度条中显示进度文本。

## 实例 096 渐变颜色的进度条

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\096

## 实例说明

在使用进度条时, 除了用文字显示进度以外, 也可以通过渐变色来显示。本实例实现了一个渐

变颜色的进度条, 实例运行结果如图 2.55 所示。

## 技术要点

在进度条控件中显示渐变色比较简单, 只需要在进度条控件的 `OnPaint` 方法中使用循环控制颜色, 然后使用 `FillRect` 填充区域就可以了。

`FillRect` 方法用指定的画刷填充区域, 语法如下:

```
void FillRect(LPCRECT lpRect,CBrush* pBrush);
```

参数说明:

- `lpRect`: 指向 `RECT` 结构的指针, 包含被填充的矩形的逻辑坐标, 可以为该参数传递 `CRect` 对象。
- `pBrush`: 标识填充矩形的画刷。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 从 `CProgressCtrl` 类派生一个子类 `CColorProgress`。
- (3) 向对话框中添加一个进度条控件, 设置控件的 `Border` 和 `Smooth` 属性, 关联变量 `m_Progress`, 其类型为 `CColorProgress`。

(4) 处理进度条的 `WM_PAINT` 消息, 在其消息处理函数中绘制进度条的文本和当前进度。代码如下:

```
void CColorProgress::OnPaint()
{
 PAINTSTRUCT ps;
 CDC* pDC = BeginPaint(&ps); //开始绘制
 int nPos = GetPos(); //获取当前进度条的位置
 CRect clientRC;
 GetClientRect(clientRC); //获取客户区域
 pDC->SetBkMode(TRANSPARENT); //将设备上下文的背景模式设置为透明
 int nMin, nMax;
 GetRange(nMin, nMax); //获取进度条的显示范围
 //获取单位刻度
 double dFraction = (double)clientRC.Width() / (nMax-nMin);
 int nLeft = nPos * dFraction; //计算左边距
 CRect leftRC = clientRC;
 leftRC.right = nLeft;
 CRect rightRC = clientRC;
 rightRC.left = nLeft;
 //以渐变色填充区域
 for(int m=255;m>0;m--)
 {
 int x,y;
 x = leftRC.Width() * m / 255;
 pDC->FillRect(CRect(0,0,x,leftRC.Height()),&CBrush(RGB(255,m,0)));
 }
 pDC->FillRect(rightRC,&CBrush(RGB(255, 255, 255))); //使用白色标识剩余的部分
 ReleaseDC(pDC); //释放设备上下文
 EndPaint(&ps); //结束窗口绘制
}
```

(5) 处理主窗口的 `WM_TIMER` 消息, 在该消息的处理函数中设置进度条的显示进度。代码如下:

```
void CProgressDlg::OnTimer(UINT nIDEvent)
{
 int nCurPos = m_Progress.GetPos(); //获取进度条的当前位置
 m_Progress.SetPos(nCurPos+1); //设置进度条的位置
 CDialog::OnTimer(nIDEvent);
}
```



图 2.55 渐变颜色的进度条

## 举一反三

根据本实例, 读者可以:

- 自绘进度条控件。

## 2.11 工具提示控件典型实例

## 实例 097 应用工具提示控件

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\097

## 实例说明

使用工具提示控件可以为控件添加提示, 工具提示控件是一个弹出窗口, 可以通过一行文本描述应用程序中的一个控件功能, 本实例就使用工具提示为控件添加提示窗口, 效果如图 2.56 所示。

## 技术要点

在使用工具提示控件时, 需要使用 Create、SetDelayTime、SetMaxTipWidth、AddTool、RelayEvent 和 UpdateTipText 方法。

(1) Create 方法。Create 方法可以创建工具提示控件窗口。语法如下:

```
BOOL Create(CWnd* pParentWnd, DWORD dwStyle = 0);
```

参数说明:

- pParentWnd: 设置工具提示控件的父窗口。
- dwStyle: 设置工具提示控件的风格, 其中, 工具提示控件的两种特定类风格如下。
  - TTS\_ALWAYSSTIP: 当鼠标停留在工具上时, 不管工具提示窗口所属的主窗口是否处于活动状态, 都显示工具提示窗口。
  - TTS\_NOPREFIX: 禁止系统将&字符从字符串中去掉。

(2) SetDelayTime 方法。SetDelayTime 方法可以为工具提示控件设置延迟时间。语法如下:

```
void SetDelayTime(UINT nDelay);
void SetDelayTime(DWORD dwDuration, int iTime);
```

参数说明:

- nDelay: 以毫秒表示延迟时间。
- dwDuration: 要获取某一段持续时间值的标志。
- iTime: 以毫秒表示指定延迟时间。

(3) SetMaxTipWidth 方法。SetMaxTipWidth 方法设置工具提示窗口的最大宽度。语法如下:

```
int SetMaxTipWidth(int iWidth);
```

参数说明:

- iWidth: 工具提示窗口的宽度。

(4) AddTool 方法。AddTool 方法用于向工具提示控件注册一个工具, 当鼠标停留在该工具上时, 工具提示控件中的信息就会显示出来。语法如下:

```
BOOL AddTool(CWnd* pWnd, UINT nIDText, LPCRECT lpRectTool = NULL, UINT nIDTool = 0);
BOOL AddTool(CWnd* pWnd, LPCWSTR lpszText = LPSTR_TEXTCALLBACK, LPCRECT lpRectTool = NULL, UINT nIDTool = 0);
```

参数说明:

- pWnd: 指向包含此工具窗口的指针。
- nIDText: 包含工具文本的字符串资源 ID。
- lpRectTool: 一个指向 RECT 结构的指针, 该结构包含工具的边界矩形坐标。
- nIDTool: 工具 ID。
- lpszText: 设置的工具文本。

(5) RelayEvent 方法。RelayEvent 方法可以将鼠标消息传递给工具提示控件。语法如下:

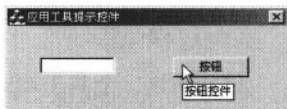


图 2.56 应用工具提示控件



void RelayEvent( LPMSG lpMsg );

参数说明:

● lpMsg: 包含要传递消息的 MSG 结构指针。

(6) UpdateTipText 方法。UpdateTipText 方法用于为工具设置提示文本。语法如下:

void UpdateTipText( LPCWSTR lpszText, CWnd\* pWnd, UINT nIDTool = 0 );

void UpdateTipText( UINT nIDText, CWnd\* pWnd, UINT nIDTool = 0 );

参数说明:

● lpszText: 设置的工具文本。

● pWnd: 包含工具的窗口指针。

● nIDTool: 工具的 ID。

● nIDText: 包含工具文本的字符串资源 ID。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 向对话框中添加一个静态文本控件、一个编辑框控件、一个组合框控件和一个按钮控件。

(3) 在对话框头文件中声明一个工具提示控件对象 m\_ToolTip。

(4) 在对话框的 OnInitDialog 函数中创建工具栏对象, 并注册工具信息, 代码如下:

```
m_ToolTip.Create(this); //创建工具提示控件
m_ToolTip.SetDelayTime(1000); //设置延迟时间
m_ToolTip.SetMaxTipWidth(300); //设置工具提示窗口的最大宽度
m_ToolTip.AddTool(GetDlgItem(IDC_EDIT1), ""); //注册编辑框控件
m_ToolTip.AddTool(GetDlgItem(IDC_BUTTON1), "按钮控件"); //注册按钮控件
```

(5) 为对话框添加 PreTranslateMessage 虚方法, 在该虚方法中设置鼠标消息的传递, 并设置编辑框的提示信息, 代码如下:

```
BOOL CToolTipControlDlg::PreTranslateMessage(MSG* pMsg) //虚方法
{
 m_ToolTip.RelayEvent(pMsg); //设置鼠标消息传递给提示控件
 m_ToolTip.UpdateTipText("编辑框控件", GetDlgItem(IDC_EDIT1)); //设置编辑框的提示文本
 return CDialog::PreTranslateMessage(pMsg); //调用基类的方法
}
```

## 举一反三

根据本实例, 读者可以:

● 自定义工具提示控件。

## 2.12 滑块控件典型实例

### 实例 098 使用滑块控件设置颜色值

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\098

## 实例说明

在使用软件时, 经常可以看到滑块控件, 本实例实现了使用滑块控件设置颜色值的功能, 运行程序, 拖动滑块, 程序将根据滑块位置对应的颜色值来绘制颜色, 程序运行结果如图 2.57 所示。

## 技术要点

在使用滑块控件时, 首先要设置控件的范

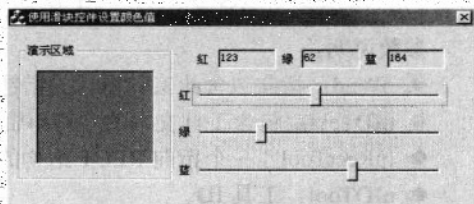


图 2.57 使用滑块控件设置颜色值

围, 然后根据拖动滑块的位置获得相应的数据, 要实现这些功能需要使用 `SetRange` 方法和 `GetPos` 方法。

(1) `SetRange` 方法。`SetRange` 方法用来设置一个滑块控件的滑块的范围(位置的最小值和最大值)。

```
void SetRange(int nMin, int nMax, BOOL bRedraw = FALSE);
```

参数说明:

`nMin`: 滑块的最小位置。

`nMax`: 滑块的最大位置。

`bRedraw`: 重画标志。如果这个参数是 `TRUE`, 则在范围被重新设置之后滑块被重画; 否则不重画滑块。

(2) `GetPos` 方法。`GetPos` 方法用来获取一个滑块控件中的滑块的当前位置。

```
int GetPos() const;
```

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 向对话框中添加群组控件、图片控件、静态文本控件、编辑框控件和滑块控件, 并设置编辑框控件为只读, 并为控件添加成员变量。

(3) 在对话框初始化时设置滑块控件的范围, 代码如下:

```
BOOL CSliderDlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 // ...系统代码省略
 //设置滑块控件区域
 m_Red.SetRange(0,255,TRUE);
 m_Green.SetRange(0,255,TRUE);
 m_Blue.SetRange(0,255,TRUE);
 return TRUE;
}
```

(4) 添加 `DrawColor`, 该函数用于在图片控件上绘制颜色, 代码如下:

```
void CSliderDlg::DrawColor()
{
 CDC* pDC = m_Color.GetDC();
 CRect rect;
 m_Color.GetClientRect(rect);
 CBrush brush(RGB(m_Red.GetPos(), m_Green.GetPos(), m_Blue.GetPos()));
 pDC->FillRect(rect, &brush);
}
```

(5) 处理滑块控件中滑块移动的响应事件, 代码如下:

```
void CSliderDlg::OnReleasedcaptureSlider1(NMHDR* pNMHDR, LRESULT* pResult)
{
 m_rEdit = m_Red.GetPos(); //获得滑块位置
 DrawColor(); //绘制颜色
 UpdateData(FALSE);
 *pResult = 0;
}

void CSliderDlg::OnReleasedcaptureSlider2(NMHDR* pNMHDR, LRESULT* pResult)
{
 m_gEdit = m_Green.GetPos(); //获得滑块位置
 DrawColor(); //绘制颜色
 UpdateData(FALSE);
 *pResult = 0;
}

void CSliderDlg::OnReleasedcaptureSlider3(NMHDR* pNMHDR, LRESULT* pResult)
{
 m_bEdit = m_Blue.GetPos(); //获得滑块位置
 DrawColor(); //绘制颜色
 UpdateData(FALSE);
 *pResult = 0;
}
```

## 举一反三

根据本实例,读者可以:

- 使用滑块控件设置窗体透明度。

## 实例 099 绘制滑块控件

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\099

## 实例说明

在一些媒体播放器软件中,通常使用滑块控件来显示当前的播放进度。这是因为滑块控件不仅可以显示进度,还可以设置播放进度。本实例实现了滑块控件的自绘,实例运行结果如图2.58所示。

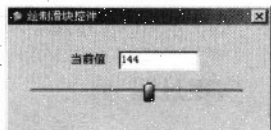


图 2.58 绘制滑块控件

## 技术要点

设计滑块控件比较简单,只需要在滑块控件的 OnPaint 方法中获取滑块客户区域,获取拖动块的区域,在这两个区域中绘制位图就可以了。由于在 OnPaint 方法中需要进行两次绘图,为了放置出现屏幕闪烁,笔者利用内存画布来实现绘图操作。将所有的绘图操作在内存画布上进行,最后将内存画布的内容输出到窗口中。内存画布的设计代码如下:

```
class CMemDC : public CDC
{
private:
 CBitmap* m_bitmap;
 CBitmap* m_oldbmp;
 CDC* m_pDC;
 CRect m_Rect;
public:
 CMemDC(CDC* pDC, const CRect& rect) : CDC() //构造函数
 {
 CreateCompatibleDC(pDC); //创建一个兼容的设备上下文
 m_bitmap = new CBitmap; //创建一个位图对象
 m_bitmap->CreateCompatibleBitmap(pDC, rect.Width(), rect.Height()); //创建一个兼容的位图
 m_oldbmp = SelectObject(m_bitmap); //选中位图
 m_pDC = pDC; //记录源设备上下文
 m_Rect = rect; //记录区域
 }
 ~CMemDC() //析构函数
 {
 m_pDC->BitBlt(m_Rect.left, m_Rect.top, m_Rect.Width(), m_Rect.Height(),
 this, m_Rect.left, m_Rect.top, SRCCOPY); //将设备上下文复制到目标上下文中
 SelectObject(m_oldbmp);
 if (m_bitmap != NULL)
 delete m_bitmap; //释放位图对象
 }
};
```

## 实现过程

- (1) 新建一个基于对话框的应用程序,在对话框中添加 CStatic 控件。
- (2) 向对话框中放置一个编辑框控件和一个滑块控件,为滑块控件关联变量,其类型为 CDrawSlider。

(3) 从 CSliderCtrl 类派生一个子类——CDrawSlider,向该类中添加成员变量。代码如下:

```
BOOL m_bBtnDown;
```

//鼠标按钮是否按下

(4) 处理滑块控件的鼠标按钮按下时的单击事件,开始鼠标捕捉。代码如下:

```
void CDrawSlider::OnLButtonDown(UINT nFlags, CPoint point)
```

```
{
 SetCapture(); //捕获鼠标
 m_bBtnDown = TRUE;
 CSliderCtrl::OnLButtonDown(nFlags, point);
}
```



(5) 处理滑块控件鼠标按钮释放时的事件，释放鼠标捕捉。代码如下：

```
void CDrawSlider::OnLButtonUp(UINT nFlags, CPoint point)
{
 ReleaseCapture(); //释放鼠标
 m_bBtnDown = FALSE;
 CSliderCtrl::OnLButtonUp(nFlags, point);
}
```

(6) 处理滑块控件的 WM\_PAINT 消息，使用位图绘制滑块的客户区域和拖动块的区域。

代码如下：

```
void CDrawSlider::OnPaint()
{
 CPaintDC dc(this);
 int nPos = GetPos(); //获取滑块的当前位置
 SetPos(nPos);
 CRect ThumbRC;
 GetThumbRect(ThumbRC); //获取滑块区域
 CBitmap bmp; //定义位图对象
 bmp.LoadBitmap(IDB_THUMB); //加载位图
 BITMAP bmpInfo; //定义位图信息
 bmp.GetBitmap(&bmpInfo);
 int bmpWidth = bmpInfo.bmWidth; //获取位图高度
 int bmpHeight = bmpInfo.bmHeight; //获取位图的高度
 CRect ClientRC, ChaneIRC;
 GetClientRect(ClientRC); //获取客户区域
 GetChannelRect(ChaneIRC); //获取通道区域
 //绘制背景
 CBitmap bmpBK; //定义位图对象
 BITMAP BKInfo; //定义位图信息
 bmpBK.LoadBitmap(IDB_SLIDERBK); //加载背景位图
 bmpBK.GetBitmap(&BKInfo); //获取位图信息
 int nBKWidth = BKInfo.bmWidth; //获取位图的宽度
 int nBKHeight = BKInfo.bmHeight; //获取位图的高度
 CDC memDC;
 memDC.CreateCompatibleDC(&dc); //创建一个兼容的设备上下文
 memDC.SelectObject(&bmpBK); //选中位图对象
 int nRightMargin = ClientRC.Width() - ChaneIRC.left - ChaneIRC.Width();
 ClientRC.right = ChaneIRC.left + ChaneIRC.Width() + nRightMargin;
 ClientRC.left = ChaneIRC.left;
 CMemDC bkMemDC(&dc, ClientRC);
 //绘制滑块的背景
 bkMemDC.StretchBlt(ChaneIRC.left, 0, ChaneIRC.Width() + ChaneIRC.left, ClientRC.Height(),
 &memDC, 0, 0, nBKWidth, nBKHeight, SRCCOPY);
 bmpBK.DeleteObject(); //删除位图对象
 memDC.DeleteDC();
 memDC.CreateCompatibleDC(&dc);
 memDC.SelectObject(&bmp); //选中滑块位图
 //绘制滑块
 bkMemDC.StretchBlt(ThumbRC.left, ThumbRC.top, ThumbRC.Width(), ClientRC.Height(),
 &memDC, 0, 0, bmpWidth, bmpHeight, SRCCOPY);
 bmp.DeleteObject(); //释放位图对象
 memDC.DeleteDC();
}
```

(7) 处理对话框水平滚动条滚动时的消息，如果是由于滑块控件引发的消息，则获取滑块控件的当前位置，将数据显示在编辑框中。这样，当用户拖动滑块时，在编辑框中显示滑块的位置。代码如下：

```
void CDrawSliderDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
 if (pScrollBar == (CScrollBar*)&m_Slider) //如果为滑块控件触发的消息
 {
 m_nRed = m_Slider.GetPos();
 UpdateData(FALSE);
 }
 CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

## 举一反三

根据本实例，读者可以：

- 设计不同样式的滑块控件。



## 2.13 标签控件典型实例

标签控件 (Tab Control) 提供了一组水平标签, 可以单击不同的标签进入相应的页面。但标签控件不能直接在各个标签页上插入控件, 只能在选中不同标签页时显示不同的对话框或控件。

### 实例 100 应用标签控件

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\100

#### 实例说明

在程序中经常会使用标签控件来切换界面。本实例实现了使用标签控件进行切换的功能, 运行结果如图 2.59 所示。

#### 技术要点

在使用标签控件时, 最主要的功能就是, 选择不同的标签时有不同的显示信息, 要实现这一功能, 需要使用标签控件的 TCN\_SELCHANGE 事件, 该事件当选中标签改变后触发。可以在该事件的处理函数中使用 GetCurSel 方法和 SetCurSel 方法获得或设置当前被选中的标签索引。

(1) GetCurSel 方法。GetCurSel 方法用于获得当前被选中的标签索引, 语法如下:

```
int GetCurSel() const;
```

返回值: 返回当前被选中的标签项索引。

(2) SetCurSel 方法。SetCurSel 方法用于将某个标签设置为当前选中的标签。语法如下:

```
int SetCurSel(int nItem);
```

参数说明:

- nItem: 标识标签索引。

返回值: 之前选中的标签索引。

#### 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 添加 3 个对话框资源, 资源 ID 分别为 IDD\_DIALOG\_EMP、IDD\_DIALOG\_CLI 和 IDD\_DIALOG\_PRO, 并设置对话框资源的 Style 属性为 Child, Border 属性为 None, 为 3 个对话框资源关联类, 分别为 CEmployee、CClient 和 Cprovidedlg, 并分别向对话框资源中添加控件。

(3) 向主对话框中添加一个标签控件, 并为标签控件关联变量 m\_Tab。

(4) 在主对话框头文件中声明 1 个图像列表对象和 3 个对话框类对象, 代码如下:

```
CImageList m_ImageList; //图像列表对象
CEmployee* m_eDlg; //员工对话框对象
CClient* m_cDlg; //客户对话框对象
CProvidedlg* m_pDlg; //供应商对话框对象
```

(5) 在主对话框的 OnInitDialog 函数中创建图像列表, 向图像列表中添加图标, 关联标签控件和图像列表控件, 并创建对话框, 代码如下:

```
m_ImageList.Create(24,24,ILC_COLOR24,ILC_MASK,1,0); //创建图像列表
m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON1)); //向图像列表中添加图标
m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON2)); //向图像列表中添加图标
m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON3)); //向图像列表中添加图标
m_Tab.SetImageList(&m_ImageList); //将图像列表关联到标签控件中
m_Tab.InsertItem(0,"员工信息",0); //插入标签项
```

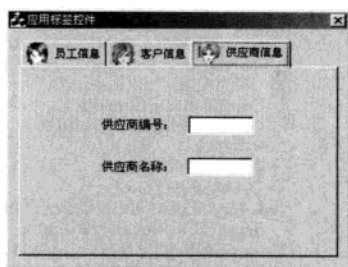


图 2.59 应用标签控件

```

m_Tab.InsertItem(1,"客户信息",1); //插入标签项
m_Tab.InsertItem(2,"供应商信息",2); //插入标签项
m_eDlg = new CEmployee; //为指针分配内存空间
m_cDlg = new CClient; //为指针分配内存空间
m_pDlg = new CProvidedlg; //为指针分配内存空间
m_eDlg->Create(IDD_DIALOG_CLI,&m_Tab); //创建员工对话框
m_cDlg->Create(IDD_DIALOG_EMP,&m_Tab); //创建客户对话框
m_pDlg->Create(IDD_DIALOG_PRO,&m_Tab); //创建供应商对话框
m_eDlg->CenterWindow(); //设置员工窗口在中心位置
m_eDlg->ShowWindow(SW_SHOW); //显示客户窗口

```

(6) 处理对话框的 TCN\_SELCHANGE 事件, 在该事件中获得当前选中标签项的索引, 根据索引判断显示的对话框, 代码如下:

```

void CUseTabDlg::OnSelchangeTab1(NMHDR* pNMHDR, LRESULT* pResult) //事件处理函数
{
 int index = m_Tab.GetCurSel(); //获得当前选中标签项索引
 switch(index) //判断标签项索引值
 {
 case 0: //值为0时
 m_eDlg->CenterWindow(); //设置员工对话框在中心位置
 m_eDlg->ShowWindow(SW_SHOW); //显示员工对话框
 m_cDlg->ShowWindow(SW_HIDE); //隐藏客户对话框
 m_pDlg->ShowWindow(SW_HIDE); //隐藏供应商对话框
 break;
 case 1: //值为1时
 m_cDlg->CenterWindow(); //设置客户对话框在中心位置
 m_eDlg->ShowWindow(SW_HIDE); //隐藏员工对话框
 m_cDlg->ShowWindow(SW_SHOW); //显示客户对话框
 m_pDlg->ShowWindow(SW_HIDE); //隐藏供应商对话框
 break;
 case 2: //值为2时
 m_pDlg->CenterWindow(); //设置供应商对话框在中心位置
 m_eDlg->ShowWindow(SW_HIDE); //隐藏员工对话框
 m_cDlg->ShowWindow(SW_HIDE); //隐藏客户对话框
 m_pDlg->ShowWindow(SW_SHOW); //显示供应商对话框
 break;
 }
 *pResult = 0;
}

```

(7) 处理对话框的 WM\_CLOSE 事件, 在主对话框关闭时销毁 3 个非模式对话框, 并释放指针, 代码如下:

```

void CUseTabDlg::OnClose()
{
 m_eDlg->DestroyWindow(); //销毁员工对话框
 delete m_eDlg; //释放员工对话框指针
 m_cDlg->DestroyWindow(); //销毁客户对话框
 delete m_cDlg; //释放客户对话框指针
 m_pDlg->DestroyWindow(); //销毁供应商对话框
 delete m_pDlg; //释放供应商对话框指针
 CDialog::OnClose(); //关闭对话框
}

```

## 举一反三

根据本实例, 读者可以:

- 使用标签控件控制显示的控件。

## 实例 101 自定义标签控件

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\02\101

### 实例说明

许多应用软件都具有漂亮的标签控件, 本实例通过自绘标签控件实现了一个类似.NET 环境下的标签控件, 实例运行结果如图 2.60 所示。

### 技术要点

在设计自定义选项卡时, 主要应用了设备上下文 CDC 类的几个方法实现区域的填充、文

本绘制等功能。下面逐一介绍。

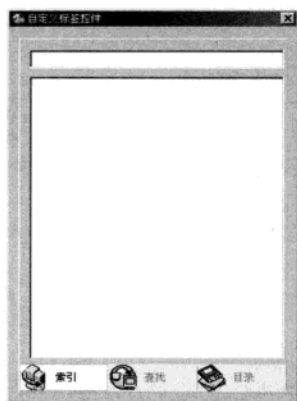


图 2.60 自定义标签控件

(1) FillSolidRect 方法。设备上下文的 FillSolidRect 方法用于使用指定的颜色来填充矩形区域。语法如下：

```
void FillSolidRect(LPCRECT lpRect, COLORREF clr);
```

参数说明：

- lpRect：表示填充的矩形区域。
- clr：表示使用的 RGB 颜色值。

(2) SetTextColor 方法。设备上下文的 SetTextColor 方法用于设置输出的文本颜色。语法如下：

```
virtual COLORREF SetTextColor(COLORREF crColor);
```

参数说明：

- crColor：表示设置的文本颜色。

返回值：表示之前设备上下文的文本颜色。

(3) SetBkMode 方法。设备上下文的 SetBkMode 方法用于设置背景模式。语法如下：

```
int SetBkMode(int nBkMode);
```

- nBkMode：表示设置的背景模式，为 OPAQUE，表示使用当前的背景颜色填充背景。为 TRANSPARENT，表示使用透明的背景模式。

返回值：表示之前设备上下文的背景模式。

(4) SelectObject 方法。设备上下文的 SelectObject 方法用于选中一个 GDI 对象。例如，当用户需要输出文本时，可以定义一个字体对象，然后利用该方法载入字体，之后输出的文本将以载入的字体输出。语法如下：

```
CGdiObject* SelectObject(CGdiObject* pObject);
```

- pObject：表示一个 GDI 对象，可以是画刷、画笔、字体、位图等对象。

返回值：表示设备上下文之前选中的 GDI 对象。

(5) DrawText 方法。设备上下文的 DrawText 方法用于按指定的格式输入文本。语法如下：

```
int DrawText(const CString& str, LPRECT lpRect, UINT nFormat);
```

- str：表示输出的字符串。
- lpRect：表示输出字符串所在的矩形区域。
- nFormat：表示文本输出格式，是一组标记的集合。为 DT\_LEFT，表示水平居左对齐，为 DT\_VCENTER，表示垂直居中，为 DT\_SINGLELINE，表示单行输出，不回车。

## 实现过程

- (1) 新建一个基于对话框的应用程序。

(2) 以 CTabCtrl 类为基类派生一个子类 CNetTabCtrl。

(3) 向对话框中添加一个标签控件，设置标签控件的 Bottom 属性，添加一个 CNetTabCtrl 类的成员变量 m\_Tab。

(4) 向 CNetTabCtrl 中添加 DrawTabBorder 方法，绘制标签页的边框。代码如下：

```
void CNetTabCtrl::DrawTabBorder(CDC *pDC, CRect &TabRC)
{
 pDC->Draw3dRect(&TabRC, RGB(255, 255, 255), RGB(177, 174, 162)); //绘制标签控件的矩形边框
}
```

(5) 向 CNetTabCtrl 中添加 DrawItemFrame 方法，绘制标签控件中各个选项卡的边框。代码如下：

```
void CNetTabCtrl::DrawItemFrame(DRAWITEMSTRUCT *lpDrawItem)
{
 int nCurSel = GetCurSel();
 int nIndex = lpDrawItem->itemID;
 BOOL bSelected = (nCurSel == nIndex);
 CDC dc;
 CRect itemRC = lpDrawItem->rcItem;
 dc.Attach(lpDrawItem->hDC);
 if (bSelected)
 {
 itemRC.OffsetRect(-1, 0);
 dc.FrameRect(&itemRC, &CBrush(RGB(172, 168, 153))); //绘制选项卡边框
 }
 else
 {
 //绘制左右两端的分隔线
 if (nIndex == 0)
 {
 dc.FillSolidRect(itemRC.left, itemRC.top + 2, 1, itemRC.Height() - 4, RGB(172, 168, 153));
 dc.FillSolidRect(itemRC.right - 1, itemRC.top + 2, 1, itemRC.Height() - 4, RGB(172, 168, 153));
 }
 dc.Detach();
 }
}
```

(6) 改写标签控件的 DrawItem 虚方法，根据当前选项卡的不同状态，绘制不同效果的选项卡。代码如下：

```
void CNetTabCtrl::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
 ASSERT(lpDrawItemStruct->CtlType == ODT_TAB);
 CDC dc;
 dc.Attach(lpDrawItemStruct->hDC);
 CRect itemRC(lpDrawItemStruct->rcItem);
 int nIndex = lpDrawItemStruct->itemID;
 int nState = lpDrawItemStruct->itemState;
 static COLORREF clrBK = RGB(229, 229, 215);
 static COLORREF clrSelBK = RGB(252, 252, 254);
 COLORREF clrText = RGB(113, 111, 100);
 if (nState & ODS_SELECTED)
 {
 dc.FillSolidRect((CRect)lpDrawItemStruct->rcItem, clrSelBK);
 clrText = RGB(0, 0, 0);
 }
 else
 {
 if (nIndex == 0)
 {
 CRect itemRC = lpDrawItemStruct->rcItem;
 itemRC.OffsetRect(-1, 0);
 dc.FillSolidRect(itemRC, clrBK);
 }
 else
 {
 dc.FillSolidRect((CRect)lpDrawItemStruct->rcItem, clrBK);
 }
 }
 //绘制项目文本和图像
 char szText[MAX_PATH] = {0};
 TC_ITEM tcltem;
 tcltem.mask = TCIF_TEXT | TCIF_IMAGE;
 tcltem.pszText = szText;
 tcltem.cchTextMax = MAX_PATH;
 GetItem(nIndex, &tcltem);
 CImageList* plmages = GetImageList();
 //定义字符数组
 //定义选项卡对象
 //设置掩码
 //获取选项卡信息
 //获取控件关联的图像列表
```



```

if (pImages != NULL)
{
 CPoint iconPT;
 iconPT.x = itemRC.left;
 iconPT.y = itemRC.top;
 pImages->Draw(&dc, tcltem.iImage, iconPT, ILD_NORMAL); //绘制选项卡显示的位图
}
itemRC.left += 40; //设置位图与文本的间距
dc.SetTextColor(clrText); //设置文本颜色
dc.SetBkMode(TRANSPARENT); //设置透明的背景模式
CFont* pFont = GetFont(); //获取字体对象
dc.SelectObject(pFont); //选中字体对象
//输出文本
dc.DrawText(szText, strlen(szText), itemRC, DT_LEFT|DT_VCENTER|DT_SINGLELINE);
dc.Detach(); //分离设备上下文句柄
}

```

(7) 处理标签控件的 WM\_PAINT 消息, 在其消息处理函数中绘制每个选项卡。代码如下:

```

void CNetTabCtrl::OnPaint()
{
 CPaintDC dc(this);
 DRAWITEMSTRUCT drawItem; //定义一个选项结构
 drawItem.CtlType = ODT_TAB; //设置选项类型为选项卡
 drawItem.CtlID = GetDlgCtrlID(); //获取标签控件的ID
 drawItem.hwndItem = GetSafeHwnd(); //获取标签控件句柄
 drawItem.hDC = dc.GetSafeHdc(); //获取设备上下文
 drawItem.itemAction = ODA_DRAWENTIRE; //设置自绘风格
 GetClientRect(&drawItem.rcItem); //获取客户区域
 CRect clientRC;
 CRect pageRC;
 pageRC = drawItem.rcItem;
 AdjustRect(FALSE, pageRC); //调整区域
 drawItem.rcItem.top = pageRC.top - 2;
 DrawTabBorder(&dc, (CRect)drawItem.rcItem); //绘制标签页的边框
 int nItemCount = GetItemCount(); //获取选项卡数量
 int nCurSel = GetCurSel(); //获取选中的选项卡索引
 if (nItemCount < 1)
 return;
 for(int i=0; i<nItemCount; i++) //遍历选项卡
 {
 drawItem.itemID = i;
 if (i == nCurSel) //当前选项卡处于选中状态
 drawItem.itemState = ODS_SELECTED; //设置选中标记
 else
 drawItem.itemState = 0;
 GetItemRect(i, &drawItem.rcItem); //获取选项卡的区域
 DrawItem(&drawItem); //调用DrawItem方法绘制选项卡
 DrawItemFrame(&drawItem); //绘制项目边框
 }
}

```

(8) 改写标签控件的 PreSubclassWindow 方法, 设置标签控件选项卡最小宽度和默认的高度和高度。代码如下:

```

void CNetTabCtrl::PreSubclassWindow()
{
 SetMinTabWidth(100); //设置最小选项卡的宽度
 SetItemSize(CSize(120, 32)); //设置选项卡的宽度和高度
 CTabCtrl::PreSubclassWindow();
}

```

(9) 创建一个对话框类——CIndexDlg, 向对话框中添加编辑框和图片控件。设置对话框 Style 属性为 Child, 设置 Border 属性为 None。

(10) 在主对话框中利用类向导为标签控件命名为 “m\_Tab”, 其类型为 CNetTabCtrl。向主对话框类中添加成员变量。代码如下:

```

CImageList m_ImageList; //定义图像列表
CIndexDlg m_IndexDlg; //定义对话框

```

(11) 在主对话框初始化时创建子对话框, 创建图像列表, 向图像列表中添加位图, 向标签控件中添加选项卡。代码如下:

```

m_IndexDlg.Create(IDD_INDEXDLG_DIALOG, &m_Tab); //在标签控件上创建子窗口
m_ImageList.Create(32, 32, ILC_COLOR16|ILC_MASK, 1, 1); //创建图像列表控件
CBitmap bmp; //定义位图对象
bmp.LoadBitmap(IDB_BITMAP1); //载入位图
m_ImageList.Add(&bmp, RGB(255, 255, 255)); //向图像列表中添加位图

```

```

bmp.Detach();
bmp.LoadBitmap(IDB_BITMAP2);
m_ImageList.Add(&bmp, RGB(255, 255, 255));
bmp.Detach();
bmp.LoadBitmap(IDB_BITMAP3);
m_ImageList.Add(&bmp, RGB(255, 255, 255));
bmp.Detach();
m_Tab.SetImageList(&m_ImageList);
m_Tab.InsertItem(0, "索引", 0);
m_Tab.InsertItem(1, "查找", 1);
m_Tab.InsertItem(2, "目录", 2);
m_Tab.SetCurSel(0);
LRESULT result;
OnSelchangeTab1(NULL, &result);

```

```

//分离位图句柄
//载入位图
//向图像列表中添加位图
//分离位图句柄
//载入位图
//向图像列表中添加位图
//分离位图句柄
//为标签控件设置选项卡
//向标签控件中插入选项卡

```

```
//将第一个选项卡选中
```

```
//当前选项卡发生了改变
```

### 举一反三

根据本实例，读者可以：

- 设计具有热点效果的标签控件。

## 2.14 控件数组典型实例

在设计程序界面时，经常会在对话框中放置多个相同的控件，例如 CStatic 控件或 CEdit 控件。如果用户需要同时访问多个控件，例如清空所有编辑框中的文本，采用控件名称逐一进行清空未免有些麻烦。MFC 应用程序完全支持控件数组，如果将相同类型的控件放置在控件数组中，那么通过遍历数组中的元素就可以访问每一个控件了。本节将通过几个实例介绍控件数组在应用程序开发中的应用。

### 实例 102 向窗体中动态添加控件

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\02\102

#### 实例说明

在使用标签控件 CTabCtrl 设计应用程序时，经常需要动态地创建控件，因为在设计期间，无法将其他控件直接放置在 CTabCtrl 控件上。本实例实现了动态向窗体中添加控件的功能，运行结果如图 2.61 所示。

#### 技术要点

动态创建控件可以分为 3 个过程，首先定义一个控件对象，然后调用控件的 Create 方法创建控件窗口，最后调用 ShowWindow 方法显示窗口。例如，下面的代码动态创建了一个编辑框，代码如下：

```

CEdit m_Edit;
m_Edit.Create(WS_BORDER, CRect(10, 10, 50, 30), this, 0); //创建编辑框控件
m_Edit.ShowWindow(SW_SHOW); //显示控件

```

由于本实例需要动态创建多个控件，为了方便管理，采用控件数组动态创建控件。定义控件数组与定义普通的数组一样，只是数组类型是控件类型。例如：

```

CStatic m_statics[6];
CStatic m_static2[6];
CEdit m_edits1[6];
CEdit m_edits2[6];

```

当创建数组控件窗口时，利用循环就可以了，代码如下：

```

for (i = 0; i < 6; i++)
{

```

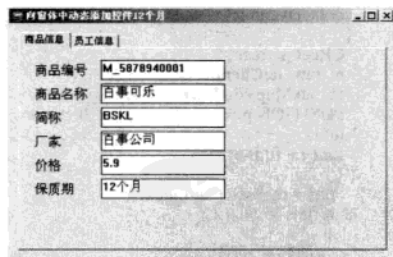


图 2.61 向窗体中动态添加控件

```

pos = m_list1.FindIndex(i);
if (pos != NULL)
{
 //创建控件窗口
 m_spagel[i].Create(m_list1.GetAt(pos), WS_CHILD, CRect(20, (i+1)*30+10, 100, (i+1)*30+30), &m_tab);
 //显示控件
 m_spagel[i].ShowWindow(SW_SHOW);
}
m_epagel[i].Create(WS_BORDER, CRect(m_rect.left+100, m_rect.top+(i+1)*30+5,
 m_rect.left+250, m_rect.top+(i+1)*30+30), this, IDE_EDITCHANGE);
m_epagel[i].ShowWindow(SW_SHOW);
}

```

在动态创建控件时,有时还需要处理控件中的各种事件。例如,处理在编辑框文本改变时的事件实现某些功能,可以通过消息映射来处理控件中的各种事件。以编辑框的文本改变的事件为例,首先在对话框中定义一个事件处理函数,并在函数的声明处使用 `afx_msg` 关键字,标识该函数是消息处理函数。例如:

```
afx_msg void OnChangeEdit1();
```

然后在资源文件中定义一个资源 ID, 例如:

```
#define IDE_EDITCHANGE 3201
```

最后在对话框的消息映射部分添加消息映射宏:

```
ON_EN_CHANGE(IDE_EDITCHANGE, OnChangeEdit1)
```

创建编辑框,将编辑框的 ID 指定为 `IDE_EDITCHANGE`, 这样在编辑框的文本发生改变时,会调用 `OnChangeEdit1` 消息处理函数。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框中添加 `CTabCtrl` 控件。

(3) 在对话框类中添加如下成员变量:

```

CStatic m_statics[6]; //标签1中的静态控件
CStatic m_static2[6]; //标签2中的静态控件
CEdit m_edits1[6]; //标签1中的编辑框
CEdit m_edits2[6]; //标签2中的编辑框
CStatic* m_spagel, *m_spagel2;
CEdit* m_epagel, *m_epagel2;
CStringList m_list1, m_list; //记录静态控件文本

```

(4) 在对话框类中添加 `CreateControl` 方法,创建控件,代码如下:

```

void CDynamicCreateDlg::CreateControl()
{
 CRect m_rect;
 m_tab.GetClientRect(m_rect);
 m_tab.MapWindowPoints(this, m_rect);
 POSITION pos = m_list1.GetTailPosition();
 int i;
 for (i = 0; i < 6; i++)
 {
 pos = m_list1.FindIndex(i);
 if (pos != NULL)
 {
 //创建控件窗口
 m_spagel[i].Create(m_list1.GetAt(pos), WS_CHILD,
 CRect(20, (i+1)*30+10, 100, (i+1)*30+30), &m_tab);
 //显示控件
 m_spagel[i].ShowWindow(SW_SHOW);
 }
 m_epagel[i].Create(WS_BORDER, CRect(m_rect.left+100, m_rect.top+(i+1)*30+5,
 m_rect.left+250, m_rect.top+(i+1)*30+30), this, IDE_EDITCHANGE); //创建控件
 m_epagel[i].ShowWindow(SW_SHOW); //显示控件
 }
 for (i = 0; i < 6; i++)
 {
 pos = m_list.FindIndex(i);
 if (pos != NULL)
 {
 m_spagel2[i].Create(m_list.GetAt(pos), WS_CHILD, CRect(20, (i+1)*30+10, 100, (i+1)*30+30), &m_tab);
 }
 m_epagel2[i].Create(WS_BORDER, CRect(m_rect.left+100, m_rect.top+(i+1)*30+5,

```

```
m_rect.left+250,m_rect.top+(i+1)*30+30,this,4554);
```

### 举一反三

根据本实例，读者可以：

- 设计共享对话框基类。

## 实例 103 公交线路模拟

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\02\103

### 实例说明

在开发有关地理定位的应用程序时，程序中经常需要描述各个地点的位置以及交通路线情况。为了清晰、准确地描述某个地点信息，通常采用图形和文字相结合的方式。本实例实现了一个公交线路的模拟程序，利用 CStatic 控件数组存储各个站点信息，实例运行结果如图 2.62 所示。

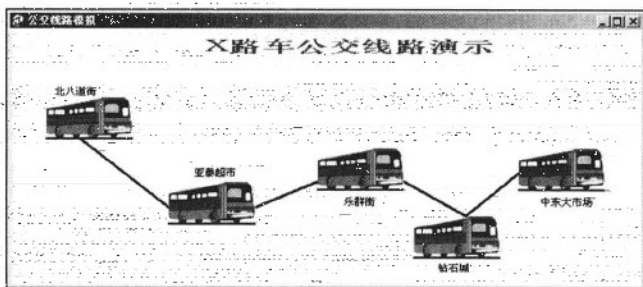


图 2.62 公交线路模拟

### 技术要点

为了简化程序，本实例利用 CStatic 控件显示图像。在程序中定义一个 CStatic 类型的控件数组，在程序初始化时调用 CStatic 的 Create 方法创建窗口，并指定窗口显示的大小和位置，然后调用 SetBitmap 方法设置 CStatic 控件显示的图像。

下面主要介绍一下 CStatic 的 Create 方法和 SetBitmap 方法。

(1) Create 方法。该方法用于创建窗口资源，语法如下：

```
BOOL Create(LPCTSTR lpszText, DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID = 0xffff);
```

参数说明：

- lpszText：用于指定控件显示的文本，本实例中，CStatic 控件需要显示位图，所以该参数应该设置为空。
- dwStyle：指定控件风格，如果控件显示位图，该参数需要包含 SS\_BITMAP 风格。
- rect：用于指定控件显示的区域。
- pParentWnd：标识控件的父窗口。
- nID：用于指定控件 ID。

(2) SetBitmap 方法。该方法用于显示位图信息，语法如下：

```
HBITMAP SetBitmap(HBITMAP hBitmap);
```

参数说明：

- hBitmap：绘制的位图句柄，通常可以利用 LoadBitmap 方法根据程序电导入的位图 ID



获取位图句柄。

- 返回值：是之前控件关联的位图句柄。如果控件之前没有关联位图，返回值为 NULL。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框类中添加 CStatic 控件数组 m\_buses。
- (3) 在对话框类中添加 DemoCircuitry 方法，创建 CStatic 控件，并设置 CStatic 控件显示的位图，代码如下：

```
void CRutineDemoDlg::DemoCircuitry()
{
 m_buses[0].Create("", WS_CHILD|SS_BITMAP, CRect(40, 80, 100, 140), this);
 m_buses[1].Create("", WS_CHILD|SS_BITMAP, CRect(180, 180, 240, 240), this);
 m_buses[2].Create("", WS_CHILD|SS_BITMAP, CRect(350, 140, 400, 380), this);
 m_buses[3].Create("", WS_CHILD|SS_BITMAP, CRect(460, 220, 500, 400), this);
 m_buses[4].Create("", WS_CHILD|SS_BITMAP|WS_EX_TRANSPARENT,
 CRect(580, 140, 650, 160), this);
 for (int i = 0; i < 5; i++)
 {
 m_buses[i].SetBitmap(LoadBitmap(AfxGetInstanceHandle(),
 MAKEINTRESOURCE(IDB_BITMAP1)));
 m_buses[i].ShowWindow(SW_SHOW);
 }
}
```

- (4) 利用属性对话框修改 CStatic 的属性，使其包含 Simple 风格。处理对话框的 WM\_CTLCOLOR 消息，使 CStatic 控件背景透明，使对话框的背景呈白色，代码如下：

```
HBRUSH CRutineDemoDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
 HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
 if (nCtlColor == CTL_COLOR_STATIC) //判断是否为CStatic控件
 {
 pDC->SetBkMode(TRANSPARENT); //设置背景透明
 }
 else
 {
 CBrush m_brush(RGB(255, 255, 255)); //创建白色画刷
 CRect m_rect;
 GetClientRect(m_rect); //获得客户区域
 pDC->SelectObject(&m_brush); //选入画刷
 pDC->FillRect(m_rect, &m_brush); //使对话框背景为白色
 return m_brush;
 }
}
```

## 举一反三

根据本实例，读者可以：

- 设计旅游线路多媒体演示程序。

## 第3章

# 图形技术

- 绘制图形
- 图像预览
- 图片效果
- 图片颜色转换
- 图形转换与缩放
- 图像的剪切、合成与识别
- 图像字体
- 图像管理
- 图片动画
- 简单游戏设计
- OpenGL 程序设计
- GDI+ 程序设计

Visual C++

## 3.1 绘制图形

利用程序自动绘制图形可以增强图形的可视度,特别是对于一些多媒体教学之类的软件来说,用程序自动绘制并控制图形无疑是一个新的突破。本节通过几个应用实例介绍绘图方面的相关知识。

### 实例 104 绘制正弦曲线

本实例是一个图形绘制的程序

实例位置: 光盘\mingrisoft\03\104

#### 实例说明

在多媒体教学时,经常需要动态地画出各种曲线进行教学演示,以加深学生对知识的理解。本实例实现的是绘制一个正弦曲线。运行程序,系统将自动绘制正弦曲线,效果如图 3.1 所示。

#### 技术要点

首先通过 SetViewportOrg 函数设置坐标原点,然后使用 MoveTo 和 LineTo 函数绘制坐标轴和曲线。下面介绍这两个函数。

(1) MoveTo 函数。该函数用于设置画笔位置。

函数原型如下:

```
CPoint MoveTo(int x, int y);
```

参数说明:

- x: 横坐标。
- y: 纵坐标。

(2) LineTo 函数。该函数用当前画笔画一条线,从当前位置连到一个指定的点。

```
BOOL LineTo(int x, int y);
```

参数说明:

- x: 横坐标。
- y: 纵坐标。

#### 实现过程

(1) 新建一个基于单文档的应用程序。

(2) 主要程序代码如下:

```
O void CSinusoidView::OnDraw(CDC* pDC)
{
 CSinusoidDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 //建立画笔
 CPen cpen,pen;
 pen.CreatePen(PS_SOLID,4,RGB(0,0,0));
 cpen.CreatePen(PS_SOLID,2,RGB(0,0,255));
 pDC->SelectObject(&cpen);
 //指定原点
 pDC->SetViewportOrg(100,245);
 pDC->SetTextColor(RGB(255,0,0));
 //绘制横坐标
 CString sPIText[]={"-1/2π","","1/2π","π","3/2π","2π","5/2π","3π","7/2π","4π","9/2π","5π"};
 for(int n=-1,nTmp=0;nTmp<=660;n++,nTmp+=60)
 {
 pDC->LineTo(60*n,0);
 pDC->LineTo(60*n,-5);
 pDC->MoveTo(60*n,0);
```

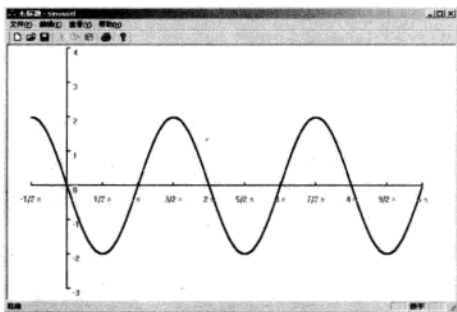


图 3.1 绘制正弦曲线

```

pDC->TextOut(60*n-sPIText[n+1].GetLength()*3,16,sPIText[n+1]);
}
pDC->MoveTo(0,0);
CString sTmp;
//绘制纵坐标
for(n=-4,nTmp=0;nTmp<=180;n++,nTmp=60*n)
{
 pDC->LineTo(0,60*n);
 pDC->LineTo(5,60*n);
 pDC->MoveTo(0,60*n);
 sTmp.Format("%d",-n);
 pDC->TextOut(10,60*n,sTmp);
}
double y,radian;
pDC->SelectObject(&pen);
for(int x=-60;x<600;x++)
{
 //弧度=x坐标/曲线宽度*角系数*π
 //y坐标=振幅*曲线宽度*sin(弧度)
 radian =x/((double)60*2)*PI;
 y=sin(radian)*2*60;
 pDC->MoveTo((int)x,(int)y);
 pDC->LineTo((int)x,(int)y);
}
cpen.DeleteObject();
pen.DeleteObject();
}

```

### 举一反三

根据本实例，读者可以：

- 绘制余弦曲线；
- 绘制双曲线。

## 实例 105 绘制蜗牛曲线

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\03\105

### 实例说明

蜗牛线是一种常用的曲线，本实例演示的是如何在程序中绘制蜗牛曲线。运行程序，程序将自动绘制蜗牛曲线，效果如图 3.2 所示。

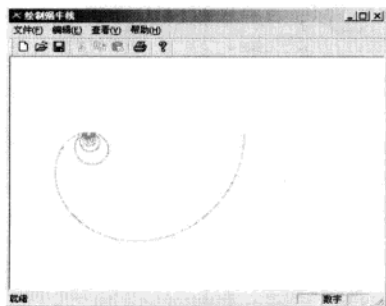


图 3.2 绘制蜗牛曲线

### 技术要点

通过 SetPixel 函数来绘制曲线上的点来实现绘制蜗牛曲线。SetPixel 函数的语法如下：

```

COLORREF SetPixel(int x, int y, COLORREF crColor);
COLORREF SetPixel(POINT point, COLORREF crColor);

```

参数说明：

- x, y：指定点的横坐标和纵坐标。



- point: 指定点的坐标。可以为该参数传递 POINT 结构或 CPoint 对象。
- crColor: 绘制点的颜色。

## 实现过程

- (1) 新建一个基于单文档的应用程序。
- (2) 主要程序代码如下:

```
void CSnailView::OnDraw(CDC* pDC)
{
 CSnailDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 pDC->SetWindowOrg(-200,-200); //设置坐标原点
 //绘制蜗牛曲线
 for(double i=0;i<40;i+=pi/600)
 {
 if(i==0)
 pDC->SetPixel(0,0,RGB(0,128,128));
 else
 pDC->SetPixel((20*sin(i)/i*cos(i))*10,(20*sin(i)/i*sin(i))*10,RGB(128,128,128));
 }
}
```

## 举一反三

根据本实例,读者可以:

- 设置点的颜色。

## 实例 106 绘制贝塞尔曲线

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\03\106

## 实例说明

贝塞尔曲线是一种常见的曲线,本实例演示的是绘制水平和垂直两个方向的贝塞尔曲线。运行程序,程序自动绘制贝塞尔曲线,效果如图 3.3 所示。

## 技术要点

CDC 类的 PolyBezier 成员函数是专门用来绘制贝塞尔曲线。该函数的语法如下:

```
BOOL PolyBezier(const POINT* lpPoints, int nCount);
```

参数说明:

- lpPoints: 含有曲线控制点和端点的 POINT 数据结构或 CPoint 对象。
- nCount: lpPoints 数组中点的数目。其值应为曲线数目的 3 倍以上,因为每条 Bezier 曲线需要 2 个控制点和 1 个终点,初始曲线还需要 1 个起点。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 主要程序代码如下:

```
void CDrawBezierDlg::OnPaint()
{
 if (IsIconic())
 {
 //...此处代码省略
 dc.DrawIcon(x, y, m_hIcon);
 }
 else
 {
 CPaintDC dc(this);
```

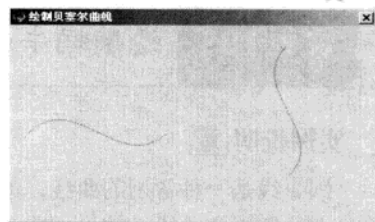


图 3.3 绘制贝塞尔曲线

```

CPen newpen;
newpen.CreatePen(PS_SOLID,1,RGB(255,128,0));//创建画笔
dc.SelectObject(&newpen);
//垂直
POINT ptv[4];
ptv[0].x=300;
ptv[0].y=20;
ptv[1].x=250;
ptv[1].y=70;
ptv[2].x=350;
ptv[2].y=120;
ptv[3].x=300;
ptv[3].y=170;
dc.PolyBezier(ptv,4); //绘制垂直贝塞尔曲线
//水平
POINT pth[4];
pth[0].x=20;
pth[0].y=120;
pth[1].x=70;
pth[1].y=70;
pth[2].x=120;
pth[2].y=170;
pth[3].x=170;
pth[3].y=120;
dc.PolyBezier(pth,4); //绘制水平贝塞尔曲线

CDialog::OnPaint();
}

```

### 举一反三

根据本实例，读者可以：

- 绘制雕刻效果。

## 实例 107 画图程序

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\03\107

### 实例说明

本实例实现了一个简单的画图功能。运行程序，在“绘图”菜单下选择要绘制的图形之后，在窗体中拖动鼠标就可以绘制想要的图形，效果如图 3.4 所示。

### 技术要点

实现画圆和矩形，需要使用 CDC 对象的 Ellipse 方法和 Rectangle 方法，而自定义画线是通过线段来记录鼠标的移动过程，这些功能都是在 WM\_MOUSEMOVE 消息中实现的。下面介绍 CDC 对象的 Ellipse 方法和 Rectangle 方法。

(1) Ellipse 方法。该方法用于画椭圆形。

语法如下：

```
BOOL Ellipse(int x1, int y1, int x2, int y2);
```

参数说明：

- x1：起始点横坐标。
- y1：起始点纵坐标。
- x2：结束点横坐标。
- y2：结束点纵坐标。

(2) Rectangle 方法。该方法用于绘制长方形。

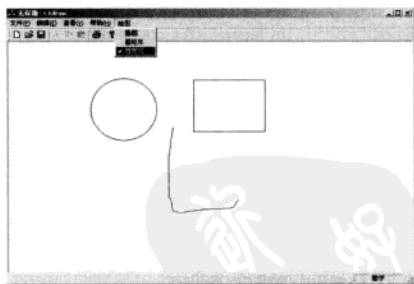


图 3.4 画图程序

语法如下:

```
BOOL Rectangle(int x1, int y1, int x2, int y2);
```

参数说明:

- x1: 左上角横坐标。
- y1: 左上角纵坐标。
- x2: 右下角横坐标。
- y2: 右下角纵坐标。

注意: WM\_MOUSEMOVE 消息在第2章已经介绍过了, 这里不再赘述。

## 实现过程

- (1) 新建一个单文档的应用程序。
- (2) 编辑菜单资源, 添加一个“绘图”菜单项, 在其下添加“画圆”、“画矩形”和“自定义”3个子菜单。
- (3) 通过类向导为新建的3个子菜单添加消息响应函数。
- (4) 主要程序代码如下:

```
void CCTDrawView::OnMouseMove(UINT nFlags, CPoint point)
{
 CClientDC dc(this);
 if(m_draw&&(nFlags&&MK_LBUTTON)) //如果是画线
 {
 dc.MoveTo(m_start);
 dc.LineTo(point);
 m_start = point;
 }
 if(m_juxing&&(nFlags&&MK_LBUTTON)) //如果是画矩形
 {
 CGdiObject* object = dc.SelectStockObject(NULL_BRUSH);
 int mdoe = dc.GetROP2();
 dc.SetROP2(R2_NOTCOPYPEN);
 dc.Rectangle(m_end.x, m_end.y, m_start.x, m_start.y);
 dc.SetROP2(mdoe);
 dc.Rectangle(m_start.x, m_start.y, point.x, point.y);
 dc.SelectObject(object);
 m_end = point;
 }
 if(m_yuan&&(nFlags&&MK_LBUTTON)) //如果是画圆
 {
 CGdiObject* object = dc.SelectStockObject(NULL_BRUSH);
 int mdoe = dc.GetROP2();
 dc.SetROP2(R2_NOTCOPYPEN);
 dc.Ellipse(m_end.x, m_end.y, m_start.x, m_start.y);
 dc.SetROP2(mdoe);
 dc.Ellipse(m_start.x, m_start.y, point.x, point.y);
 dc.SelectObject(object);
 m_end = point;
 }
 CView::OnMouseMove(nFlags, point);
}
```

## 举一反三

根据本实例, 读者可以:

- 编写画板软件。

## 实例 108 绘制立体模型

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\03\108

## 实例说明

立体几何图形在多媒体教学中经常会用到, 本实例演示的是如何动态地绘制一个正方体图案。运行程序, 在编辑框中分别输入长、宽、高和角度, 单击“绘图”按钮将绘制立体模型,

效果如图 3.5 所示。

### 技术要点

首先设置一个图形控件作为画板, 通过 `SetViewportOrg` 函数获得中心点坐标, 根据输入的数据计算立体模型的各个点坐标, 使用 CDC 对象的 `Rectangle` 方法绘制正面的长方形, 根据角度判断哪几条边用虚线绘制。

注意: CDC 对象的 `Rectangle` 方法参照实例 107 中的内容。

### 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 向窗体中添加 1 个图片控件, 设置 `Type` 属性为 `Rectangle` 和 `Color` 属性为 `white`, 再添加 4 个编辑框控件。

(3) 主要程序代码如下:

```
void CDrawcubeDlg::OnButDraw()
{
 CDC* pDC;
 pDC = m_palette.GetDC(); //获得设备上下文
 CRect rc, rect;
 m_palette.GetClientRect(&rect); //获得控件客户区域
 m_palette.GetWindowRect(&rc); //获得控件窗口区域
 pDC->FillRect(rect, NULL); //填充区域
 //取出中心点
 CPoint center;
 center.x = rc.Width()/2;
 center.y = rc.Height()/2;
 pDC->SetViewportOrg(center); //设置原点
 CString slength, swidth, sheight, sangle;
 //获得编辑框中数据
 m_length.GetWindowText(slength);
 m_width.GetWindowText(swidth);
 m_height.GetWindowText(sheight);
 m_angle.GetWindowText(sangle);
 int nlength, nwidth, nheight, nangle;
 //将编辑框中数据转换为整型数据
 nlength = atoi(slength);
 nwidth = atoi(swidth);
 nheight = atoi(sheight);
 nangle = atoi(sangle);
 CPoint LTop, LBottom, RTop, RBottom;
 LTop.x = 1 - nlength/2;
 LTop.y = 1 - nheight/2;
 RTop.x = nlength/2;
 RTop.y = 1 - nheight/2;
 LBottom.x = 1 - nlength/2;
 LBottom.y = nheight/2;
 RBottom.x = nlength/2;
 RBottom.y = nheight/2;
 CPen pen(PS_SOLID, 1, RGB(0, 0, 0)); //创建画笔
 CPen DOTPen;
 DOTPen.CreatePen(PS_DOT, 1, RGB(0, 0, 0)); //虚线画笔
 pDC->SelectObject(&pen);
 //画正面矩形
 pDC->Rectangle(LTop.x, LTop.y, RBottom.x, RBottom.y);
 CPoint LeftTop, RightTop;
 LeftTop.x = (long)(LTop.x + (cos(nangle*PI/180)*nwidth));
 LeftTop.y = (long)(LTop.y - (sin(nangle*PI/180)*nwidth));
 RightTop.x = LeftTop.x + nlength;
 RightTop.y = LeftTop.y;
 pDC->MoveTo(LTop);
 pDC->LineTo(LeftTop);
 pDC->LineTo(RightTop);
 pDC->LineTo(RTop);
 CPoint Other, DotPoint;
```

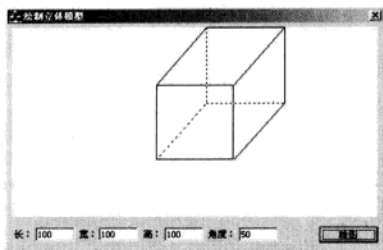


图 3.5 绘制立体模型



```
DotPoint.x=LeftTop.x;
DotPoint.y=LeftTop.y+nheight;
pDC->MoveTo(RightTop);
if(nangle<89) //画立方体其他面实边线
{
 pDC->SelectObject(&pen);
 Other.x=RightTop.x;
 Other.y=RightTop.y+nheight;
 pDC->LineTo(Other);
 pDC->LineTo(RBottom);
 pDC->SelectObject(&DOTPen);
 pDC->MoveTo(LeftTop);
 pDC->LineTo(DotPoint);
 pDC->LineTo(LBottom);
}
else //绘制矩形背面虚线
{
 pDC->SelectObject(&DOTPen);
 Other.x=RightTop.x;
 Other.y=RightTop.y+nheight;
 pDC->LineTo(Other);
 pDC->LineTo(RBottom);
 pDC->SelectObject(&pen);
 pDC->MoveTo(LeftTop);
 pDC->LineTo(DotPoint);
 pDC->LineTo(LBottom);
}
pDC->SelectObject(&DOTPen);
pDC->MoveTo(DotPoint);
pDC->LineTo(Other);
}
```

### 举一反三

根据本实例，读者可以：

● 绘制多边形。

## 实例 109

## 利用 IFS 算法绘制自然景物

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\03\109

### 实例说明

本实例通过 IFS 算法来实现对自然景物的模拟，IFS 可以模拟的景物很多，有山、树、三叶草和皇冠等，本实例实现对山的模拟，如图 3.6 所示。

### 技术要点

IFS(Iterator Function System)是一种分形几何系统，主要通过仿射坐标变换来生成几何图形，仿射坐标变换是旋转、扭曲和平移 3 种效果的叠加。

### 实现过程

- (1) 创建名为 IFS 的单文档 MFC 工程。
- (2) 在头文件中加入变量的声明，具体参照代码。
- (3) 在 CIFSView 类的头文件中声明变量：

```
class CIFSView : public CView
{
public:
 CIFSDoc* GetDocument();
 //声明的变量
 double x,y;int i,k;
 int MaxY;
 double a[8],b[8],c[8],d[8],e[8],f[8],g[8],h[8];
 COLORREF m_pColor;
```

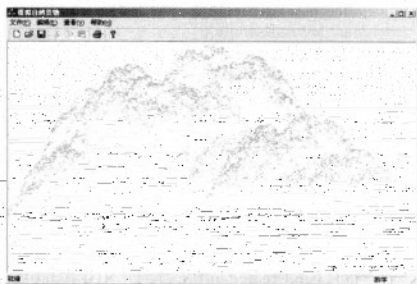


图 3.6 利用 IFS 算法绘制自然景物

```
int drawtrue;
int m_N;
int stepx,stepy;
int totalsteps;
}
```

(4) 在实现文件中加入常量声明:

```
#define MaxY 400
```

(5) 在构造函数中实现对变量的初始化:

```
CIFSView::CIFSView()
```

```
{
 m_pColor=RGB(0,255,0);
 m_N=4;
 stepx=3;
 stepy=3;
 totalsteps=32000;
}
```

(6) 在 OnDraw 函数中进行图形的绘制, 代码如下:

```
void CIFSView::OnDraw(CDC* pDC)
```

```
{
 CIFSDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 for(i=1;i<m_N;i++)
 {
 p[i]=p[i]+p[i-1];
 }
 float xj,m;
 x=0;y=0;
 srand(unsigned(time(NULL))); //设置随机数种子
 for(i=0;i<totalsteps;i++)
 {
 m=float(rand()); //获得随机数
 xj=float(m/RAND_MAX);
 if(xj<=p[0]) k=0;
 if((xj>p[0])&&(xj<=p[1])) k=1;
 if((xj>p[1])&&(xj<=p[2])) k=2;
 if((xj>p[2])&&(xj<=p[3])) k=3;
 if((xj>p[3])&&(xj<=p[4])) k=4;
 if((xj>p[4])&&(xj<=p[5])) k=5;
 if((xj>p[5])&&(xj<=p[6])) k=6;
 if((xj>p[6]) &&(xj<=p[7])) k=7;
 x=a[k]*x+b[k]*y+e[k];
 y=c[k]*x+d[k]*y+f[k];
 if(i>10)
 pDC->SetPixel(int(MaxY*x/stepx+MaxY/2), MaxY-int(MaxY*y/stepy+30)-100,m_pColor);//绘制点
 }
}
```

(7) 在 OnInitialUpdate 中为 IFS 算法所需要的数组设置初始值, 代码如下:

```
void CIFSView::OnInitialUpdate()
```

```
{
 CView::OnInitialUpdate();
 CIFSDoc *pDoc=GetDocument();
 ASSERT_VALID(pDoc);
 //变量的初始化
 a[0]=0.7;a[1]=0.5;a[2]=-0.4;a[3]=-0.5;
 b[0]=0.0;b[1]=0;b[2]=0;b[3]=0;
 c[0]=0;c[1]=0;c[2]=1;c[3]=0;
 d[0]=0.8;d[1]=0.5;d[2]=-0.4;d[3]=0.5;
 e[0]=0;c[1]=2;e[2]=-0;c[3]=2;
 f[0]=0;f[1]=0;f[2]=1;f[3]=1;
 p[0]=0.25;p[1]=0.25;p[2]=0.25;p[3]=0.25;
}
```

## 举一反三

根据本实例, 读者可以:

- 模拟树;
- 模拟三叶草。



## 3.2 图像预览

在应用程序中经常会用到图像预览。下面的几个实例主要实现了图片的自动预览、批量浏览、浏览大幅 BMP 图片以及控制图片大小等功能。

### 实例 110 图片自动预览程序

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\03\110

#### 实例说明

可以通过自动预览的功能来浏览多幅图片，这样就不用手动去选择要浏览的图片了。运行程序，单击“文件”/“打开”菜单项，选择一幅 BMP 图片，程序将自动预览和这幅图片相同文件夹下的其他图片，效果如图 3.7 所示。

#### 技术要点

本实例使用定时器设置在一定时间后自动显示下一幅图片，在 MFC 中可以使用 SetTimer 函数来定义和开始一个定时器，通过处理消息 WM\_TIMER 来达到定时控制的功能，最后使用 KillTimer 函数关闭定时器。

SetTimer 函数原型如下：

```
UINT SetTimer(UINT nIDEvent, UINT nElapse, void (CALLBACK EXPORT* lpfnTimer)(HWND, UINT, UINT, DWORD));
```

参数说明：

- nIDEvent：定时器的标识 ID。
- nElapse：延迟时间（多长时间重复一次），单位是毫秒。
- 第 3 个参数：重复调用的函数的地址指针，为 NULL 时将发送 WM\_TIMER 消息。

KillTimer 函数原型如下：

```
BOOL KillTimer(int nIDEvent);
```

参数说明：

- nIDEvent：定时器的标识 ID。

#### 实现过程

(1) 新建一个基于单文档的应用程序。

(2) 主要程序代码如下：

```
void CBmpView::OnTimer(UINT nIDEvent)
{
 CString str=Search(GetDocument()->GetPathName()); //获得文件路径名
 if(str=="")
 return;
 AfxGetApp()->OpenDocumentFile(str); //打开文件
 CView::OnTimer(nIDEvent);
}

//自定义函数，查找并打开图片
CString CBmpView::Search(CString curstr)
{
 long handle;
 if(curstr.IsEmpty())
 return "";
 if(_getcwd(buffer, 1000)==NULL)
 {
 AfxMessageBox("没有当前路径,请打开一个图像文件!");
 return "";
 }
}
```



图 3.7 图片自动预览程序

```

CString m_sPartname;
int len = curstr.GetLength();
int i;
for(i = len-1; curstr[i] != '\\'; i--)
 m_sPartname.Insert(0, curstr[i]); //获得文件名
i++;
while(i--<0)
 buffer[i]=curstr[i];
if (_chdir(buffer) != 0)
 return "";
bool b_notfinde=false;
struct _finddata_t filestruct;
// 开始查找工作, 找到当前目录下的第一个实体(文件或子目录),
// “*”表示查找任何的文件或子目录, filestruct为查找结果
handle = _findfirst("*", &filestruct);
do{
 if((handle == -1)) // 当handle为-1时, 表示当前目录为空, 结束查找并返回
 break;
 // 检查找到的第一个实体是否是一个目录
 if (::GetFileAttributes(filestruct.name) & FILE_ATTRIBUTE_DIRECTORY)
 {
 continue;
 }
 CString Filename=filestruct.name;
 {
 CString tailstr;
 //获取文件扩展名
 tailstr = Filename.Mid(Filename.GetLength()-3);
 tailstr.MakeUpper();
 Filename.MakeUpper();
 m_sPartname.MakeUpper();
 if(tailstr=="BMP")
 {
 if(b_notfinde==false)
 {
 if(m_sPartname==Filename)
 b_notfinde=true;
 }
 else
 {
 _findclose(handle);
 return Filename;
 }
 }
 }
} while(_findnext(handle, &filestruct)==0);
_findclose(handle);
this->KillTimer(1); //关闭定时器
AfxMessageBox("已经到达最后一个图像文件!");
return "";
}

char szFilter[] = "BMP Files(*.BMP)|*.BMP";

```

### 举一反三

根据本实例, 读者可以:

- 开发电子相册程序;
- 制作屏保程序。

## 实例 111 图片批量浏览

本实例是一个提高效率的程序

实例位置: 光盘\mingrisoft\03\111

### 实例说明

很多时候, 需要通过“缩略图”来浏览某一个文件夹中的图片, 本实例即实现了与缩略



图相同的功能。运行程序，单击“打开”按钮，选择一幅 BMP 图片，程序将自动打开和这幅图片相同文件夹下的其他 3 幅图片，单击“上一条”、“下一条”、“上一组”和“下一组”按钮可以浏览该文件夹下的其他图片，效果如图 3.8 所示。

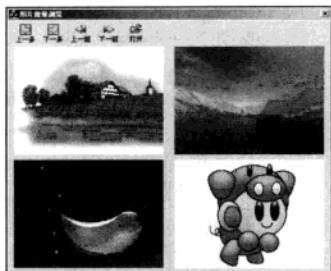


图 3.8 图片批量浏览

## 技术要点

本实例使用 LoadImage 函数装载位图文件，该函数用于装载图标、光标或位图。函数原型如下：

HANDLE LoadImage (NINSTANCE hinst, LPCTSTR lpszName, UINT uType, int cxDesired, int CyDesired, UINT fuLoad);

参数说明：

- hinst：处理包含被装载图像模块的特例。
- lpszName：处理图像装载。如果参数 hinst 为非空，而且参数 fuLoad 不包括 LR\_LOADFROMFILE 的值时，那么参数 lpszName 是一个指向保留在 hinst 模块中装载的图像资源名称，并以 NULL 为结束符的字符串。
- uType：指定被装载图像类型。此参数值如表 3.1 所示。

表 3.1 uType 参数可选值表

| 参 数          | 描 述  |
|--------------|------|
| IMAGE_BITMAP | 装载位图 |
| IMAGE_CURSOR | 装载光标 |
| IMAGE_ICON   | 装载图标 |

- cxDesired：指定图标或光标的宽度，以像素为单位。如果此参数为零并且参数 fuLoad 值为 LR\_DEFAULTSIZE，那么函数使用 SM\_CXICON 或 SM\_CXCURSOR 系统公制值设定宽度；如果此参数为零并且值 LR\_DEFAULTSIZE 没有被使用，那么函数使用目前的资源宽度。
- cyDesired：指定图标或光标的高度，以像素为单位。如果此参数为零并且参数 fuLoad 值为 LR\_DEFAULTSIZE，那么函数使用 SM\_CXICON 或 SM\_CXCURSOR 系统公制值设定高度；如果此参数为零并且值 LR\_DEFAULTSIZE 没有被使用，那么函数使用目前的资源高度。
- fuLoad：根据表 3.2 所示的复合值指定函数值。

表 3.2 fuLoad 参数可选值表

| 参 数                 | 描 述                                                                                                                       |
|---------------------|---------------------------------------------------------------------------------------------------------------------------|
| LR_DEFAULTCOLOR     | 默认标志，它不作任何事情，其含义是“无 LR_MONOCHROME”                                                                                        |
| LR_CREATEDIBSECTION | 当参数 uType 指定为 IMAGE_BITMAP 时，使得函数返回一个 DIB 部分位图，而不是一个兼容的位图                                                                 |
| LR_DEFAULTSIZE      | 若 cxDesired 或 cyDesired 未被设为零，使用系统指定的标识光标或图标的宽和高。如果这个参数不被设置且 cxDesired 或 cyDesired 被设为零，函数使用实际资源尺寸。如果资源包含多个图像，则使用第一个图像的大小 |
| LR_LOADFROMFILE     | 根据参数 lpszName 的值装载图像。若标记未被给定，lpszName 的值为资源名称                                                                             |
| LR_LOADMAP3DCOLORS  | 查找图像的颜色表并且按下面相应的 3D 颜色表的灰度进行替换                                                                                            |

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 创建一个图形控件类 CPicture，其基类为 CStatic。
- (3) 主要程序代码如下：

```

CString CBrowsebmpsDlg::Search(CString curstr,bool judge)
{
 long handle;
 if(curstr.IsEmpty()) //判断路径是否为空
 return "";
 if(_getcwd(buffer, 1000)==NULL)
 {
 AfxMessageBox("没有当前路径,请打开一个图像文件!");
 return "";
 }

 CString m_sbefore="";
 CString m_sPartname;
 int len = curstr.GetLength();
 int i;
 for(i = len-1;curstr[i] != '\\';i--)
 m_sPartname.Insert(0,curstr[i]); //获得文件名
 i++;
 while(i--<0)
 buffer[i]=curstr[i];
 if(_chdir(buffer) != 0)
 return "";

 bool b_notfinde=false;
 struct _finddata_t filestruct;
 // 开始查找工作, 找到当前目录下的第一个实体(文件或子目录)
 // "*" 表示查找任何的文件或子目录, filestruct为查找结果
 handle = _findfirst("*. ", &filestruct);
 do{
 if((handle ==-1)) // 当handle为-1时, 表示当前目录为空, 结束查找并返回
 break;
 // 检查找到的第一个实体是否是一个目录
 if (::GetFileAttributes(filestruct.name) & FILE_ATTRIBUTE_DIRECTORY)
 {
 continue ;
 }
 CString Filename=filestruct.name;
 {
 CString tailstr;
 tailstr = Filename.Mid(Filename.GetLength()-3);
 tailstr.MakeUpper();
 Filename.MakeUpper();
 m_sPartname.MakeUpper();
 if(tailstr=="BMP")
 {
 if(judge)
 {
 if(b_notfinde==false)
 {
 if(m_sPartname==Filename)
 b_notfinde=true;
 }
 else
 {
 _findclose(handle);
 return Filename;
 }
 }
 else
 {
 if(m_sPartname==Filename)
 {
 _findclose(handle);
 if(m_sbefore=="")
 {
 AfxMessageBox("已经到达第一个图像文件!");
 }
 return m_sbefore;
 }
 b_notfinde=true;
 m_sbefore = Filename;
 }
 }
 }
 } while(_findnext(handle, &filestruct)!=0);
 _findclose(handle);
 if(judge)
 {

```

```
AfxMessageBox("已经到达最后一个图像文件!");
}
else
{
 AfxMessageBox("已经到达第一个图像文件!");
}
return "";
}

char szFilter[] = "BMP Files(*.BMP)|*.BMP";

void CBrowsebmpsDlg::Loadpicture(CString str)
{
 for(int i=0;i<4;i++)
 {
 if(i != 0)
 {
 str = strText+Search(str,true);
 }
 strFile[i]=str;
 HBITMAP m_hBitmap;
 m_hBitmap=(HBITMAP)::LoadImage(AfxGetInstanceHandle(),str,IMAGE_BITMAP,0,0,
 LR_LOADFROMFILE|LR_DEFAULTCOLOR|LR_DEFAULTSIZE); //加载图片资源
 if (m_hBitmap != NULL)
 {
 picture[i].SetImage(m_hBitmap); //显示图片
 picture[i].ShowWindow(SW_SHOW); //显示控件
 }
 }
}
```

### 举一反三

根据本实例，读者可以：

- 开发图片浏览器程序；
- 浏览不同格式的图片。

## 实例 112 浏览大幅 BMP 图片

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\03\112

### 实例说明

在开发地理定位信息系统时，程序中需要处理大幅的图片，但是程序窗口通常不能够显示整张图片，因此在浏览图片时，需要使用滚动条控件。本实例实现了利用滚动条浏览大幅图片的功能，效果如图 3.9 所示。

### 技术要点

在使用滚动条控件 CScrollBar 时，需要处理以下几种情况：

- (1) 用户单击滚动条的左右或上下按钮时，设置滚动块的位置，并滚动窗口。
- (2) 用户拖动滚动块时，设置滚动块的位置，并滚动窗口。
- (3) 用户单击滚动块的左右或上下空白滚动区域时，设置滚动块的位置，并滚动窗口。

图 3.10 描述了这 3 种情况。

当用户触发了滚动条的消息时，拥有滚动条的对话框会收到 WM\_HSCROLL（水平滚动条）或 WM\_VSCROLL（垂直滚动条）消息。以水平滚动条为例，对话框将调用 OnHScroll 消息处理函数。下面分析 OnHScroll 消息处理函数：

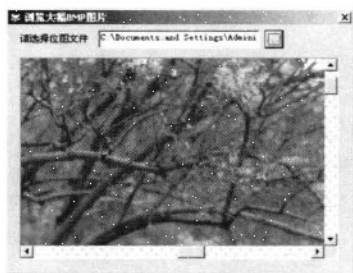


图 3.9 浏览大幅 BMP 图片

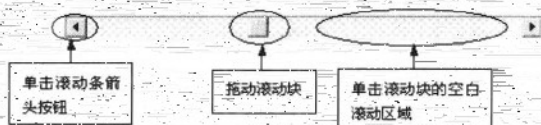


图 3.10 滚动条描述

```
void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
```

参数说明：

- **nSBCode**：标识用户触发滚动条的消息代码。可选值如下。
  - **SB\_LEFT**：表示滚动到左边缘。
  - **SB\_ENDSCROLL**：表示滚动结束。
  - **SB\_LINELEFT**：表示单击滚动条的左方按钮。
  - **SB\_LINERIGHT**：表示单击滚动条的右方按钮。
  - **SB\_PAGELEFT**：表示在滚动块的左方滚动区域按下鼠标按钮。
  - **SB\_PAGERIGHT**：表示滚动到右边缘。
  - **SB\_THUMBPOSITION**：表示结束拖动滚动块。
  - **SB\_THUMBTRACK**：表示正在拖动滚动块。
- **nPos**：标识滚动块的位置，只有在 nSBCode 为 **SB\_THUMBTRACK** 或 **SB\_THUMBPOSITION** 时才可用。
- **pScrollBar**：标识滚动条控件指针。

默认情况下，用户在触发滚动条消息时，CScrollBar 控件是不会自动调整滚动块的位置的，用户需要在滚动条的消息处理函数中根据 nSBCode 的情况设置滚动块的位置，并执行额外的动作，例如本实例需要根据滚动块的位置适当地滚动窗口，代码如下：

```
void CBmpDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
```

```
{
 int pos,min,max,thumbwidth;
 SCROLLINFO vinfo;
 GetScrollInfo(SB_HORZ,&vinfo);
 pos = vinfo.nPos;
 min = vinfo.nMin;
 max = vinfo.nMax;
 thumbwidth = vinfo.nPage;
 switch (nSBCode)
 {
 break;
 case SB_THUMBTRACK: //拖动滚动块
 ScrollWindow(-(nPos-pos),0); //滚动窗口
 SetScrollPos(SB_HORZ,nPos); //设置滚动块的位置
 break;
 case SB_LINELEFT: //单击左箭头
 SetScrollPos(SB_HORZ,pos-1);
 if (pos !=0)
 ScrollWindow(1,0);
 break;
 //代码省略
 }
 CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加图片控件、按钮控件、编辑框控件。
- (3) 新建一个对话框类 CBmpDlg，设置对话框的风格，如图 3.11 所示。
- (4) 处理 CBmpDlg 对话框的 WM\_HSCROLL 和 WM\_VSCROLL 消息，设置滚动条的位置，并适当滚动窗口，代码如下：



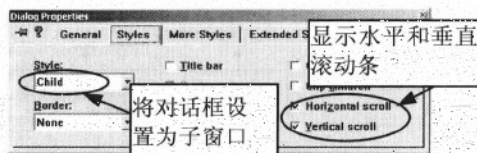


图 3.11 对话框属性设置

//WM\_HSCROLL消息处理函数

void CBmpDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar\* pScrollBar)

```
{
 int pos,min,max,thumbwidth;
 SCROLLINFO vinfo;
 GetScrollInfo(SB_HORZ,&vinfo);
 pos = vinfo.nPos;
 min = vinfo.nMin;
 max = vinfo.nMax;
 thumbwidth = vinfo.nPage;
 switch (nSBCode)
 {
 break;
 case SB_THUMBTRACK: //拖动滚动块
 ScrollWindow(-(nPos-pos),0);
 SetScrollPos(SB_HORZ,nPos);
 break;
 case SB_LINELEFT: //单击左箭头
 SetScrollPos(SB_HORZ,pos-1);
 if (pos !=0)
 ScrollWindow(1,0);
 break;
 case SB_LINERIGHT: //单击右箭头
 SetScrollPos(SB_HORZ,pos+1);
 if (pos+thumbwidth <max)
 ScrollWindow(-1,0);
 break;
 case SB_PAGELEFT: //在滚动块的左方空白滚动区域单击, 增量为6
 SetScrollPos(SB_HORZ,pos-6);
 if (pos+thumbwidth >0)
 ScrollWindow(6,0);
 break;
 case SB_PAGERIGHT: //在滚动块的右方空白滚动区域单击, 增量为6
 SetScrollPos(SB_HORZ,pos+6);
 if (pos+thumbwidth <max)
 ScrollWindow(-6,0);
 break;
 }
 CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

//WM\_VSCROLL消息处理函数

void CBmpDlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar\* pScrollBar)

```
{
 int pos,min,max,thumbwidth;
 SCROLLINFO vinfo;
 GetScrollInfo(SB_VERT,&vinfo);
 pos = vinfo.nPos;
 min = vinfo.nMin;
 max = vinfo.nMax;
 thumbwidth = vinfo.nPage;
 switch (nSBCode)
 {
 case SB_THUMBTRACK: //拖动滚动块
 pos = GetScrollPos(SB_VERT);
 ScrollWindow(0,-(nPos-pos));
 SetScrollPos(SB_VERT,nPos);
 break;
 case SB_LINELEFT: //单击左箭头
 pos = GetScrollPos(SB_VERT);
 SetScrollPos(SB_VERT,pos-1);
 if (pos !=0)
 ScrollWindow(0,1);
 break;
 case SB_LINERIGHT: //单击右箭头
 pos = GetScrollPos(SB_VERT);
 SetScrollPos(SB_VERT,pos+1);
 if (pos+thumbwidth <max)
 ScrollWindow(0,-1);
 break;
 }
}
```

```

case SB_PAGELLEFT: //在滚动块的上方空白滚动区域单击, 增量为6
 SetScrollPos(SB_VERT, pos-6);
 if (pos+thumbwidth > 0)
 ScrollWindow(0, 6);
break;
case SB_PAGERIGHT: //在滚动块的下方空白滚动区域单击, 增量为6
 SetScrollPos(SB_VERT, pos+6);
 if (pos+thumbwidth < max)
 ScrollWindow(0, -6);
break;
}
CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}

```

(5) 处理对话框 CBmpDlg 的 WM\_MOUSEWHEEL 消息, 在用户滚动鼠标滚轮时, 移动滚动条, 代码如下:

```

BOOL CBmpDlg::OnMouseWheel(UINT nFlags, short zDelta, CPoint pt)
{
 SCROLLINFO vinfo;
 GetScrollInfo(SB_VERT, &vinfo); //获取滚动条信息
 int min, max, thumbwidth;
 min = vinfo.nMin;
 max = vinfo.nMax;
 thumbwidth = vinfo.nPage;
 int pos = GetScrollPos(SB_VERT); //获得滚动块位置
 if (zDelta > 0)
 {
 if (pos == 0)
 return TRUE;
 SetScrollPos(SB_VERT, pos-6); //设置滚动块位置
 ScrollWindow(0, 6);
 }
 else
 {
 if ((pos+thumbwidth) >= max)
 return TRUE;
 SetScrollPos(SB_VERT, pos+6); //设置滚动块位置
 ScrollWindow(0, -6);
 }
 return TRUE;
}

```

(6) 在主对话框中定义一个 CBmpDlg 对象 m\_BrownDlg, 并在对话框初始化时 (OnInitDialog 方法中) 创建子对话框, 代码如下:

```

//创建子对话框
m_BrownDlg.Create(IDD_BMPDLG_DIALOG, this);
CRect rect, framerect;
m_BrownDlg.GetClientRect(rect);
m_Frame.GetClientRect(framerect); //获得图片控件的客户区域
m_Frame.MapWindowPoints(this, framerect);
m_BrownDlg.MoveWindow(framerect); //设置对话框在图片控件位置
m_BrownDlg.ShowWindow(SW_SHOW); //显示对话框

```

(7) 处理主对话框的 WM\_MOUSEWHEEL 消息, 调用子窗口 CBmpDlg 的 WM\_MOUSEWHEEL 消息处理函数移动滚动条, 代码如下:

```

BOOL CBrownBMPDlg::OnMouseWheel(UINT nFlags, short zDelta, CPoint pt)
{
 m_BrownDlg.OnMouseWheel(nFlags, zDelta, pt);
 return CDialog::OnMouseWheel(nFlags, zDelta, pt);
}

```

### 举一反三

根据本实例, 读者可以:

- 设计画图程序。

## 实例 113 放大和缩小图片

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\03\113

### 实例说明

在浏览图片的时候, 可以通过将图片放大或缩小来方便用户浏览。本实例实现的是在

窗体上通过按钮来控制图片的大小。运行程序,单击窗体左侧的按钮可以根据比例来显示图片,效果如图 3.12 所示。

### 技术要点

本实例使用 StretchBlt 函数根据设置的比例大小重画图片。

StretchBlt 函数:绘制位图文件。函数原型如下:

BOOL StretchBlt( int x, int y, int nWidth, int nHeight, CDC\* pSrcDC, int xSrc, int ySrc, int nSrcWidth, int nSrcHeight, DWORD dwRop );

参数说明:

- x: 左上角横坐标。
- y: 左上角纵坐标。
- nWidth: 显示宽度。
- nHeight: 显示高度。
- pSrcDC: 设备环境指针。
- xSrc: 复制位图左上角横坐标。
- ySrc: 复制位图左上角纵坐标。
- nSrcWidth: 源位图宽度。
- nSrcHeight: 源位图高度。
- dwRop: 光栅操作类型。

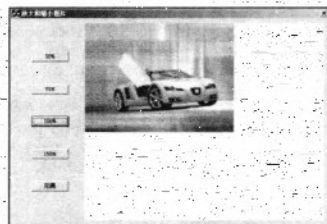


图 3.12 放大和缩小图片

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗口中添加 1 个图片控件和 5 个按钮控件。
- (3) 向程序中导入一幅位图。
- (4) 主要程序代码如下:

```
void CPictureDlg::DrawPicture(int num)
{
 //获得窗口大小
 CRect r;
 m_picture.GetClientRect(&r);
 CDC* pDC = m_picture.GetDC(); //获得设备上下文
 //填充背景
 pDC->FillRect(&r, NULL);
 //将位图选进设备场景中
 CBitmap cbmp;
 cbmp.LoadBitmap(IDB_BITMAP1); //加载图片资源
 CDC memdc;
 memdc.CreateCompatibleDC(pDC);
 memdc.SelectObject(&cbmp);
 //获得位图参数
 long width, height;
 cbmp.GetBitmap(&bmp);
 width = bmp.bmWidth;
 height = bmp.bmHeight;
 //开始缩放
 switch(num)
 {
 case 0://50%显示
 pDC->StretchBlt(r.left, r.top, (int)(width*0.5), (int)(height*0.5), &memdc, 0, 0,
 bmp.bmWidth, bmp.bmHeight, SRCCOPY);
 break;
 case 1://75%显示
 pDC->StretchBlt(r.left, r.top, (int)(width*0.75), (int)(height*0.75), &memdc, 0, 0,
 bmp.bmWidth, bmp.bmHeight, SRCCOPY);
 break;
 case 2://100%显示
 pDC->BitBlt(r.left, r.top, r.Width(), r.Height(), &memdc, 0, 0, SRCCOPY);
```

```

break;
case 3://150%显示
pDC->StretchBlt(r.left,r.top,(int)(width*1.5),(int)(height*1.5),&memdc,0,0,
bmp.bmWidth,bmp.bmHeight,SRCCOPY);
break;
case 4://充满窗口
pDC->StretchBlt(r.left,r.top,r.Width(),r.Height(),&memdc,0,0,
bmp.bmWidth,bmp.bmHeight,SRCCOPY);
break;
}
}

```

## 举一反三

根据本实例，读者可以：

- 在窗体中动态控制控件的大小。

## 实例 114 图像任意角度旋转

这是一个可以提高分析能力的实例

实例位置：光盘\mingrisoft\03\114

## 实例说明

图像旋转是图像的几何变换，是在不改变图像内容的前提下实现对像素进行空间几何变换的一种处理方式。在 GDI+ 中 Bitmap 提供了图像旋转的功能，但是它只能对固定的一些角度进行旋转，例如  $90^\circ$ 、 $180^\circ$ 、 $270^\circ$  等。在本节实例中，利用数学公式实现对图像的任意角度旋转，效果如图 3.13 所示。

## 技术要点

图像旋转核心内容就是旋转公式。有了旋转公式，将位图数据带入，就可以实现图像旋转了。下面笔者来介绍图像旋转公式的推导过程。图 3.14 描述了平面内一坐标点顺时针旋转  $\theta$  角的情况。

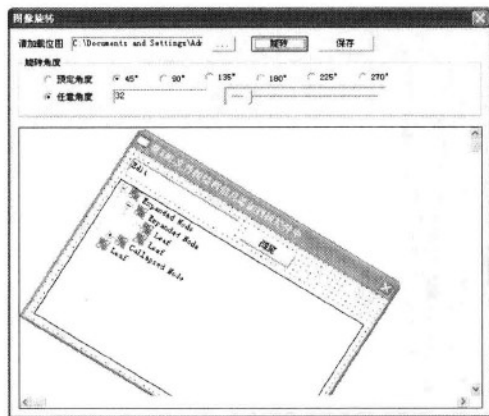


图 3.13 图像任意角度旋转

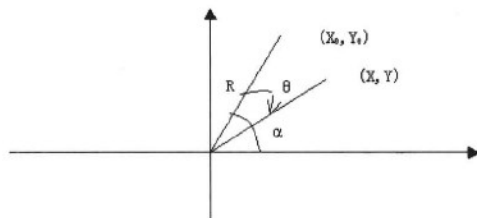


图 3.14 坐标点旋转平面图

图中旋转角度为  $\theta$  角。根据三角函数有：

$$X_0 = R \cos \alpha$$

$$Y_0 = R \sin \alpha$$

旋转后

$$X = R \cos(\alpha - \theta)$$

$$Y = R \sin(\alpha - \theta)$$

根据正弦加法定理和余弦加法定理可知：

$$\cos(\alpha - \theta) = \cos \alpha \cos \theta + \sin \alpha \sin \theta$$



$$\sin(\alpha-\theta) = \sin\alpha\cos\theta - \cos\alpha\sin\theta$$

则

$$X = R\cos\alpha\cos\theta + R\sin\alpha\sin\theta$$

$$= X_0\cos\theta + Y_0\sin\theta$$

$$Y = R\sin\alpha\cos\theta - R\cos\alpha\sin\theta$$

$$= Y_0\cos\theta - X_0\sin\theta$$

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} X_0 & Y_0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

有了上面的公式, 还需要进行逆运算。因为, 在进行位图旋转时, 在确定旋转后的位图大小之后, 可以确定旋转后的位图矩形区域, 需要访问该区域的每一个坐标, 以获得该坐标对应于源位图的坐标, 最后获得该坐标对应于源位图坐标点处的像素数据。因此需要进行逆运算。

因为

$$X = X_0\cos\theta + Y_0\sin\theta$$

有

$$Y_0 = \frac{X - X_0\cos\theta}{\sin\theta}$$

因为

$$Y = Y_0\cos\theta - X_0\sin\theta$$

有

$$Y_0 = \frac{Y + X_0\sin\theta}{\cos\theta}$$

则有

$$\frac{X - X_0\cos\theta}{\sin\theta} = \frac{Y + X_0\sin\theta}{\cos\theta}$$

将其展开有

$$X\cos\theta - X_0\cos^2\theta = Y\sin\theta + X_0\sin^2\theta$$

导出

$$X\cos\theta - Y\sin\theta = X_0\cos^2\theta + X_0\sin^2\theta$$

$$= X_0(\cos^2\theta + \sin^2\theta)$$

$$= X_0$$

即

$$X_0 = X\cos\theta - Y\sin\theta$$

按照上面的方法, 可以导出

$$Y_0 = X\sin\theta + Y\cos\theta$$

这样对应的逆运算矩阵为

$$\begin{bmatrix} X_0 & Y_0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

有了转换公式, 我们的图像转换任务并没有完成, 还需要进行坐标转换。以一幅图像为例, 在计算机中的原点为图像的左上角, 如图 3.15 所示。

而在数学中, 坐标原点为图像的中心点, 如图 3.16 所示。



图 3.15 计算机中表示的图像原点坐标



图 3.16 数学坐标系中的图像原点

假设位图的高度为  $H$ ，宽度为  $W$ ，则图像坐标  $(X_0, Y_0)$  与数学坐标  $(X, Y)$  的关系是

$$X = X_0 - 0.5W$$

$$Y = -Y_0 + 0.5H$$

对应的矩阵表示为

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} X_0 & Y_0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5W & 0.5H & 1 \end{bmatrix}$$

逆运算为

$$X_0 = X + 0.5W$$

$$Y_0 = -Y + 0.5H$$

逆运算矩阵为

$$\begin{bmatrix} X_0 & Y_0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5W & 0.5H & 1 \end{bmatrix}$$

在经过数学公式旋转后，还需要数学坐标转换为新位图的图像坐标。转换的公式为

$$X = X_0 + 0.5W'$$

$$Y = -Y_0 + 0.5H'$$

其中， $W'$  和  $H'$  表示旋转后位图的宽度和高度。

矩阵表示为

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} X_0 & Y_0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5W' & 0.5H' & 1 \end{bmatrix}$$

逆运算为

$$X_0 = X - 0.5W'$$

$$Y_0 = -Y + 0.5H'$$

逆运算矩阵为

$$\begin{bmatrix} X_0 & Y_0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5W' & 0.5H' & 1 \end{bmatrix}$$

有了上述的旋转公式，就可以进行图像旋转了。

## 实现过程

- (1) 新建一个基于对话框的工程。
- (2) 向对话框中添加按钮、编辑框、单选按钮、滑块、图片控件。
- (3) 处理“...”按钮的单击事件，首先加载位图图像，读取位图信息，然后再将位图数据

转换为4个字节表示的位图数据,便于对图像进行处理,最后根据位图的大小设置窗口的滚动范围。代码如下:

```
void CRotateImage::OnBtLoad()
{
 CFileDialog fDlg(TRUE, "", "", OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
 "位图文件|*.bmp|", this); //定义文件打开窗口
 if (fDlg.DoModal() == IDOK)
 {
 CString csFileName = fDlg.GetPathName(); //获取选择的图像名称
 m_SrcFile = fDlg.GetPathName();
 m_BmpName.SetWindowText(csFileName);
 if (m_hBmp != NULL) //判断之前是否已加载了图像
 {
 DeleteObject(m_hBmp); //删除之前加载的图像
 m_hBmp = NULL;
 }
 m_hBmp = (HBITMAP)LoadImage(NULL, csFileName, IMAGE_BITMAP, 0, 0,
 LR_LOADFROMFILE); //加载图像信息
 if (m_hBmp) //图像是否加载成功
 {
 m_Image.SetBitmap(m_hBmp); //显示图像
 m_bLoaded = TRUE; //设置加载标记为已加载
 }
 CFile file; //定义文件对象
 file.Open(csFileName, CFile::modeRead); //开发文件
 //读取位图文件头
 file.Read(&m_bmFileHeader, sizeof(BITMAPFILEHEADER));
 //读取位图信息头
 file.Read(&m_bmInfoHeader, sizeof(BITMAPINFOHEADER));
 int szPalette = 0;
 if (m_bmInfoHeader.biBitCount != 24) //判断是否为真彩色位图
 {
 file.Close();
 MessageBox("请选择真彩色位图!", "提示");
 return;
 }
 int nBmpData = m_bmInfoHeader.biSizeImage; //获取位图数据的大小
 if (m_pBmpData != NULL) //判断是否已有位图数据
 {
 delete []m_pBmpData; //删除位图数据
 m_pBmpData = NULL;
 }
 m_pBmpData = new BYTE[nBmpData]; //为位图数据分配空间
 file.ReadHuge(m_pBmpData, nBmpData); //从文件中读取位图数据
 file.Close(); //关闭文件
 //将位图数据转换为每个像素4字节形式,计算转换后的图像大小
 int sizeofbuffer = m_bmInfoHeader.biWidth * m_bmInfoHeader.biHeight * 4;
 //计算位图每行填充的字节数
 int externWidth;
 externWidth = m_bmInfoHeader.biWidth * 3;
 if (externWidth % 4 != 0)
 externWidth = 4 - externWidth % 4;
 else
 externWidth = 0;
 int k = 0;
 //为转换后的位图数据分配空间
 BYTE* m_pImageTempBuffer = new BYTE[sizeofbuffer];
 //按行遍历位图
 for (int n = m_bmInfoHeader.biHeight - 1; n >= 0; n--)
 {
 //按列遍历位图
 for (UINT m = 0; m < m_bmInfoHeader.biWidth * 3; m += 3)
 {
 //将源位图数据读取到目标位图中
 m_pImageTempBuffer[k] = m_pBmpData[n * (m_bmInfoHeader.biWidth * 3 +
 externWidth) + m];
 m_pImageTempBuffer[k + 1] = m_pBmpData[n * (m_bmInfoHeader.biWidth * 3 +
 externWidth) + m + 1];
 m_pImageTempBuffer[k + 2] = m_pBmpData[n * (m_bmInfoHeader.biWidth * 3 +
 externWidth) + m + 2];
 m_pImageTempBuffer[k + 3] = 255; //第4位填充为255
 k += 4; //填充下一个像素
 }
 }
 delete []m_pBmpData; //释放位图数据
 m_pBmpData = new BYTE[sizeofbuffer]; //重新分配空间
 memcpy(m_pBmpData, m_pImageTempBuffer, sizeofbuffer); //复制图像数据
 }
}
```

```
delete [m_pImageTempBuffer; //释放临时的图像数据

CRect bmpRC, wndRC;
m_ImagePanel.GetClientRect(wndRC); //获取面板的客户区域
m_Image.GetClientRect(bmpRC); //获取图像控件的客户区域
m_ImagePanel.OnHScroll(SB_LEFT, 1, NULL);
m_ImagePanel.OnVScroll(SB_LEFT, 1, NULL);
//设置滚动范围
m_ImagePanel.SetScrollRange(SB_VERT, 0, bmpRC.Height() - wndRC.Height());
m_ImagePanel.SetScrollRange(SB_HORZ, 0, bmpRC.Width() - wndRC.Width());
}
```

(4) 向对话框中添加 RotateBmp 方法, 按指定的角度旋转图像。其中, 参数 pBmpData 表示源位图的实际数据, pTmpData 表示缩放后的位图数据, nWidth, nHeight 表示源位图的宽度和高度, nzmWidth、nzmHeight 表示缩放后的位图宽度和高度, dXRate、dYRate 表示水平和垂直方向的缩放比例。代码如下:

```
void CRotateImage::RotateBmp(BYTE *pBmpData, BYTE *pDesData, int nWidth, int nHeight,
int nDesWidth, int nDesHeight, double dAngle)
{
 double dSin = sin(dAngle); //计算正弦值
 double dCos = cos(dAngle); //计算余弦值
 pDesData = new BYTE[nDesWidth * nDesHeight * 4]; //为目标位图数据分配内存空间
 memset(pDesData, 255, nDesWidth * nDesHeight * 4); //初始化内存空间为0
 //计算与坐标x、y无关的变量
 double dX = -0.5 * nDesWidth * dCos - 0.5 * nDesHeight * dSin + 0.5 * nWidth;
 double dY = 0.5 * nDesWidth * dSin - 0.5 * nDesHeight * dCos + 0.5 * nHeight;
 BYTE* pSrc = NULL; //定义指向源位图数据的指针
 BYTE* pDes = NULL; //定义指向目的位图数据的指针
 int x = 0;
 int y = 0;
 for (int h = 0; h < nDesHeight; h++) //按行遍历位图
 {
 for (int w = 0; w < nDesWidth; w++) //按列遍历位图
 {
 //加0.5是为了向上取整
 x = (int)(w * dCos + h * dSin + dX + 0.5); //获取旋转前的横坐标
 y = (int)(-w * dSin + h * dCos + dY + 0.5); //获取旋转前的纵坐标
 if (x == nWidth)
 {
 x--;
 }
 if (y == nHeight)
 {
 y--;
 }
 pSrc = pBmpData + y * nWidth * 4 + x * 4; //根据坐标获取源位图指定点的数据
 pDes = pDesData + h * nDesWidth * 4 + w * 4; //根据坐标获取目的位图指定点的数据
 if (x >= 0 && x < nDesWidth && y >= 0 && y < nDesHeight)
 {
 memcpy(pDes, pSrc, 4); //复制源位图数据到目的位图数据
 }
 }
 }
}
```

(5) 向对话框中添加 RotationImage 方法, 获取源图像和目的图像的信息, 并调用 RotateBmp 方法旋转图像数据。其实现代码如下:

```
void CRotateImage::RotationImage(BITMAPINFOHEADER *pBmInfo, int nDegree)
{
 UINT srcWidth = pBmInfo->biWidth; //获取源图像宽度
 UINT srcHeight = pBmInfo->biHeight; //获取源图像高度
 double pi = 3.1415926535; //定义π的值
 double dRadian = nDegree * (pi / 180.0); //将角度转换为弧度
 //计算目标图像宽度和高度
 UINT desWidth = (abs)(srcHeight * sin(dRadian)) + (abs)(srcWidth * cos(dRadian)) + 1;
 UINT desHeight = (abs)(srcHeight * cos(dRadian)) + (abs)(srcWidth * sin(dRadian)) + 1;
 UINT desLineBytes = desWidth * pBmInfo->biBitCount / 8; //计算每行字节数
 int mod = desLineBytes % 4; //计算每行填充字节数
 if (mod != 0)
 desLineBytes += 4 - mod;
 BYTE* psrcData = m_pBmpData; //获取源图像数据
 BYTE* psrcPixData = NULL;
 BYTE* pdesPixData = NULL;
 //为目标图像数据分配空间
```



```

m_hGlobal = GlobalAlloc(GMEM_MOVEABLE, desHeight * desLineBytes);
m_RotData = (BYTE *) GlobalLock(m_hGlobal); //锁定堆空间, 获取指向堆空间的指针
memset(m_RotData, 0, desHeight * desLineBytes); //初始化目标图像数据
desBmInfo.biBitCount = pBmInfo->biBitCount; //设置目标位图的颜色位数
desBmInfo.biClrImportant = pBmInfo->biClrImportant;
desBmInfo.biClrUsed = pBmInfo->biClrUsed;
desBmInfo.biCompression = pBmInfo->biCompression;
desBmInfo.biPlanes = pBmInfo->biPlanes; //设置调色板数量
desBmInfo.biSize = sizeof(BITMAPINFOHEADER); //设置位图信息头结构大小
desBmInfo.biXPelsPerMeter = pBmInfo->biXPelsPerMeter;
desBmInfo.biYPelsPerMeter = pBmInfo->biYPelsPerMeter;
BYTE * pTemp = NULL;
//调用RotateBmp方法旋转图像
RotateBmp(psrcData, pTemp, srcWidth, srcHeight, desWidth, desHeight, dRadian);
desBmInfo.biHeight = desHeight; //设置目标图像宽度
desBmInfo.biWidth = desWidth; //设置目标图像高度
desLineBytes = desWidth * pBmInfo->biBitCount / 8; //计算每行字节数
mod = desLineBytes % 4; //计算每行数据需要填充几个字节
if (mod != 0)
 desLineBytes += 4 - mod;
desBmInfo.biSizeImage = desHeight * desLineBytes; //设置图像数据大小
m_bmInfoHeader = desBmInfo;
if (m_pBmpData != NULL) //判断位图数据是否为空
{
 delete [] m_pBmpData; //释放位图数据
 m_pBmpData = NULL;
}
m_pBmpData = new BYTE[desHeight * desWidth * 4]; //重新分配空间
memset(m_pBmpData, 255, desHeight * desWidth * 4); //初始化内存空间
memcpy(m_pBmpData, pTemp, desHeight * desWidth * 4); //复制位图数据
delete [] pTemp; //删除临时的位图数据
pTemp = NULL;
}

```

(6) 处理“旋转”按钮的单击事件, 旋转已加载的位图。代码首先确定用户设置的旋转方式和旋转角度, 然后调用 RotationImage 方法旋转图像数据, 接着将旋转后的图像数据转换为 3 个字节表示像素的位图数据格式, 最后显示旋转后的位图图像。现代代码如下:

```

void CRotateImage::OnBtRotate()
{
 if (m_bLoaded)
 {
 //确定旋转方式
 CButton* pButton = (CButton*) GetDlgItem(IDC_FIXDEGREE);
 int nState = 0;
 int nDegree = 0;
 if (pButton != NULL)
 {
 nState = pButton->GetCheck(); //判断按钮是否处于选中状态
 }
 if (nState) //预定角度
 {
 //利用循环遍历单选按钮, 确定用户选择的旋转角度
 for (int nID = IDC_ROTATE45; nID <= IDC_ROTATE270; nID++)
 {
 pButton = (CButton*) GetDlgItem(nID);
 if (pButton != NULL)
 {
 nState = pButton->GetCheck();
 if (nState)
 {
 CString csText;
 pButton->GetWindowText(csText);
 int nPos = csText.Find("°");
 nDegree = atoi(csText.Left(nPos)); //获取旋转角度
 break;
 }
 }
 }
 }
 else //固定角度
 {
 UpdateData(FALSE);
 nDegree = m_nDegree; //读取旋转角度
 }
 RotationImage(&m_bmInfoHeader, nDegree); //调用RotationImage方法旋转图像
 BYTE byByteAlign; //位图行字节对齐
 UINT outHeight = m_bmInfoHeader.biHeight; //获取旋转后的位图高度
 }
}

```

```

UINT outWidth = m_bmpInfoHeader.biWidth; //获取旋转后的位图宽度
//为位图数据分配空间
BYTE* pBmpData = new BYTE [m_bmpInfoHeader.biSizeImage];
memset(pBmpData,0,m_bmpInfoHeader.biSizeImage); //初始化内存空间
//获取旋转后的图像数据(4字节表示), 指向最后一行数据
BYTE * pListData =m_pBmpData+((DWORD)outHeight-1)*outWidth*4;
//计算每行位图数据需要填充的字节数
if (outWidth %4 != 0)
 byByteAlign = 4- ((outWidth*3L) % 4);
else
 byByteAlign = 0;
BYTE byZeroData = 0;
BYTE* pTmpData = pBmpData;
for (int y=0 ;y<outHeight;y++) //按行遍历图像
{
 for (int x=0;x<outWidth;x++) //按列遍历图像
 {
 memcpy(pTmpData,pListData,3); //赋值图像数据
 pTmpData += 3; //指向下一个像素数据
 pListData += 4; //指向下一个像素数据
 }
 for (int i=0; i<byByteAlign; i++) //每行填充数据
 {
 memcpy(pTmpData,&byZeroData,1);
 pTmpData =pTmpData + 1;
 }
 pListData -= 2L*outWidth*4; //指向上一行数据
}
CDC *pDC = m_Image.GetDC();
BITMAPINFO bInfo;
bInfo.bmiHeader = m_bmpInfoHeader;
//创建并显示位图
HBITMAP hBmp = m_Image.SetBitmap(CreateDIBitmap(pDC->m_hDC,&m_bmpInfoHeader,
 CBM_INIT,pBmpData,&bInfo,DIB_RGB_COLORS));
if (hBmp != NULL) //判断之前是否显示图像
{
 ::DeleteObject(hBmp); //删除之前显示的图像
}
delete [] pBmpData; //删除位图数据
CRect bmpRC,wndRC;
m_ImagePanel.GetClientRect(wndRC); //获取面板客户区域
m_Image.GetClientRect(bmpRC); //获取图片控件客户区域
m_ImagePanel.OnHScroll(SB_LEFT, 1, NULL);
m_ImagePanel.OnVScroll(SB_LEFT, 1, NULL);
//设置滚动范围
m_ImagePanel.SetScrollRange(SB_VERT,0,bmpRC.Height()-wndRC.Height());
m_ImagePanel.SetScrollRange(SB_HORZ,0,bmpRC.Width()-wndRC.Width());
}
}

```

(7) 处理“保存”按钮的单击事件, 保存旋转后的图像。方法是将旋转后的图像数据转换为3个字节表示像素的实际位图数据, 然后根据位图信息定义位图文件头、位图信息头, 有了位图文件头、位图信息头和位图数据, 就可以生成位图文件了。代码如下:

```

void CRotateImage::OnBtSave()
{
 if (m_bLoaded) //已经加载图像
 {
 CFileDialog fDlg(FALSE,"bmp","Demo.bmp",OFN_HIDEREADONLY|
 OFN_OVERWRITEPROMPT,"位图文件|*.bmp|"); //定义文件保存对话框
 if (fDlg.DoModal()==IDOK)
 {
 try
 {
 CString csSaveName = fDlg.GetPathName(); //获取保存的图像名称
 CFile file; //定义文件对象
 //创建并打开文件
 file.Open(csSaveName,CFile::modeCreate|CFile::modeReadWrite);
 //写入位图文件信息头
 file.Write(&m_bmpFileHeader,sizeof(m_bmpFileHeader));
 //写入位图信息头
 file.Write(&m_bmpInfoHeader,sizeof(m_bmpInfoHeader));
 BYTE byByteAlign;
 UINT outHeight = m_bmpInfoHeader.biHeight; //获取位图的高度
 UINT outWidth = m_bmpInfoHeader.biWidth; //获取位图的宽度
 //指向旋转的图像数据的最后一行
 BYTE * pListData = m_pBmpData+((DWORD)outHeight-1)*outWidth*4;
 //计算需要填充的字节数
 }
 }
 }
}

```



```

if (outWidth % 4 != 0)
 byByteAlign = 4 - ((outWidth * 3L) % 4);
else
 byByteAlign = 0;
BYTE byZeroData = 0;
for (int y=0; y<outHeight; y++) //按行遍历图像
{
 for (int x=0; x<outWidth; x++) //按列遍历图像
 {
 file.Write(pListData, 3); //写入位图实际数据
 pListData += 4; //指向下一个像素
 }
 for (int i=0; i<byByteAlign; i++) //位图字节填充
 {
 file.Write(&byZeroData, 1); //写入位图实际数据
 }
 pListData -= 2L*outWidth*4; //指向下一行数据
}
file.Close(); //关闭文件
}
catch(...)
{
 MessageBox("文件保存失败!", "提示");
}
}
}

```

### 举一反三

根据本实例,读者可以:

- 实现图像的定时旋转

## 3.3 图片效果

图片特效设计丰富了计算机的内容,也为程序设计增添了不少色彩。本节主要通过图片马赛克效果、图片百叶窗效果、电影胶片特效和翻转图片效果几个实例来介绍如何对图片进行特效设计。

### 实例 115 图片马赛克效果

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\03\115

#### 实例说明

本实例实现了图片的马赛克效果,主要通过使用“编辑”/“马赛克”菜单命令,将图片以若干小块的方式逐渐显示出来,直到整张图片显示出来,如图3.17所示。

#### 技术要点

本实例主要应用 DrawDib 库进行图像的绘制,使用该库可以提高图像的显示速度,它是图像处理的常用库,库里包含许多关于图像处理的函数,使用起来比较方便。要使用 DrawDib 库必须先用 DrawDibOpen 打开 HDRAWDIB 对象,然后通过 HDRAWDIB 对象进行图像绘制,HDRAWDIB 对象是 DrawDib 库的设备环境句柄。

本实例的绘制过程是使用函数 DrawDibOpen 打开 HDRAWDIB 对象,然后使用函数 CreateDIBSection 创建与设备无关的位图,通过 BitBlt 函数将图像绘制在内存设备环境中,最后利用函数 DrawDibDraw 使用特效显示图像。

CreateDIBSection 函数是创建与设备无关的位图,语法如下:

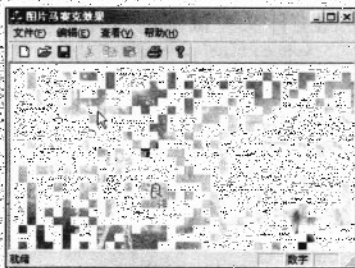


图 3.17 图片马赛克效果



```

HBITMAP CreateDIBSection(HDC hdc,CONST BITMAPINFO *pbmi,
 UINT iUsage,VOID *ppvBits, HANDLE hSection,DWORD dwOffset);

```

参数说明:

- hdc: 设备环境句柄。
- pbmi: BITMAPINFO 结构对象。
- iUsage: 说明 BITMAPINFO 结构中 bmiColors 的数据类型。
- ppvBits: 与设备无关位图数据的缓存。
- hSection: 可以创建位图对象的句柄。
- dwOffset: 位图数据在文件中的位置。

DrawDibDraw 函数是绘制与设备无关位图到屏幕上, 语法如下:

```

BOOL DrawDibDraw(HDRAWDIB hdd,HDC hdc,int xDst,int yDst,int dxDst,int dyDst,
 LPBITMAPINFOHEADER lpbi,LPVOID lpBits,int xSrc,int ySrc,int dxSrc,int dySrc,UINT wFlag);

```

参数说明:

- hdd: HDRAWDIB 对象句柄。
- hdc: 设备环境句柄。
- xDst: 目标矩形左顶点 x 轴坐标。
- yDst: 目标矩形左顶点 y 轴坐标。
- dxDst: 目标矩形的宽度。
- dyDst: 目标矩形的高度。
- lpbi: BITMAPINFOHEADER 结构对象。
- lpBits: 位图数据缓存。
- xSrc: 源矩形左顶点 x 轴坐标。
- ySrc: 源矩形左顶点 y 轴坐标。
- dxSrc: 源矩形的宽度。
- dySrc: 源矩形的高度。
- wFlag: 绘制的标识。

## 实现过程

- (1) 新建一个基于单文档的应用程序。
- (2) 在工程中加入对 vfw32.lib 的引用。
- (3) 在头文件中加入 “#include “vfw.h”” 语句。
- (4) 添加菜单项, 设置 ID 属性为 ID\_VIEW, Caption 属性为 “马赛克”, 并通过类向导添加菜单响应函数 OnView。

- (5) 在 CMosaicViewView 类的头文件中声明变量:

```

class CMosaicViewView : public CView
{
protected:
 CMosaicViewView();
 DECLARE_DYNCREATE(CMosaicViewView)
 //加入变量声明
 BITMAP bm;
 HBITMAP m_bmpSrc;
 COLORREF *m_clrSrc;
 CSize m_size;
 HDRAWDIB m_hDrawDib;
}

```

- (6) 在构造函数中初始化 HDRAWDIB 对象, 代码如下:

```

CMosaicViewView::CMosaicViewView()
{
 m_hDrawDib=NULL;
 m_hDrawDib=DrawDibOpen();
 HBITMAP hBmp=
 (HBITMAP)LoadImage(NULL,"bitmap.bmp",IMAGE_BITMAP,0,0,LR_LOADFROMFILE);//加载图片资源
}

```



```
HDC hMemDC=CreateCompatibleDC(NULL);
if(hMemDC)
{
 GetObject(hBmp,sizeof(bm),&bm);
 m_size=CSize(bm.bmWidth,bm.bmHeight);
 BITMAPINFOHEADER RGB32BITSBITMAPINFO=
 {sizeof(BITMAPINFOHEADER),bm.bmWidth,bm.bmHeight,
 1,32,BI_RGB,0,0,0,0};
 m_bmpSrc=CreateDIBSection(hMemDC,(BITMAPINFO*)&RGB32BITSBITMAPINFO,
 DIB_RGB_COLORS,(VOID**)&m_clrSrc,NULL,0);
 if(m_bmpSrc)
 {
 HBITMAP hOldBmp=(HBITMAP)SelectObject(hMemDC,m_bmpSrc);
 HDC hDC=CreateCompatibleDC(hMemDC); //创建兼容的设备上下文
 if(hDC)
 {
 HBITMAP hOldBmp2=(HBITMAP)SelectObject(hDC,hBmp); //选入图像
 BitBlt(hMemDC,0,0,bm.bmWidth,bm.bmHeight,hDC,0,0,SRCCOPY); //绘制图像
 SelectObject(hDC,hOldBmp2);
 DeleteDC(hDC);
 }
 SelectObject(hMemDC,hOldBmp);
 }
 DeleteDC(hMemDC);
}
}
```

(7) 在析构函数中实现 HDRAWDIB 对象的关闭和销毁,代码如下:

```
CMosaicViewView::~~CMosaicViewView()
{
 if(m_hDrawDib!=NULL)
 {
 DrawDibClose(m_hDrawDib);
 m_hDrawDib=NULL;
 }
 if(m_bmpSrc!=NULL)
 DeleteObject(m_bmpSrc);
}
}
```

(8) “查看” / “马赛克” 菜单命令的实现函数用于实现绘制图像,代码如下:

```
void CMosaicViewView::OnView()
{
 if(m_bmpSrc==NULL||m_hDrawDib==NULL)return;
 Invalidate();
 int nTileSize=10;
 int nTileNum=((m_size.cx+nTileSize-1)/nTileSize)*((m_size.cy+nTileSize-1)/nTileSize);
 POINT *pt=new POINT[nTileNum];
 int x=0;
 int y=0;
 int i;
 for(i=0;i<nTileNum;i++)
 {
 pt[i].x=x;
 pt[i].y=y;
 x=x+nTileSize;
 if(x>m_size.cx)
 {
 x=0;
 y=y+nTileSize;
 }
 }
 BITMAPINFOHEADER RGB32BITSBITMAPINFO=
 {sizeof(BITMAPINFOHEADER),m_size.cx,m_size.cy,1,32,BI_RGB,0,0,0,0}; //设置位图头结构
 CPaintDC dc(this);
 DrawDibRealize(m_hDrawDib,dc.GetSafeHdc(),TRUE);
 double fMax=RAND_MAX;
 for(i=nTileNum-1;i>=0;i--)
 {
 int n=(int)((double)nTileNum*rand()/fMax);
 x=pt[n].x;
 y=pt[n].y;
 DrawDibDraw(m_hDrawDib,dc.GetSafeHdc(),10+x,10+y,nTileSize,nTileSize,
 &RGB32BITSBITMAPINFO,(LPVOID)m_clrSrc,
 x,y,nTileSize,nTileSize,DDF_BACKGROUNDPALETTE);
 pt[n].x=pt[i].x;
 pt[n].y=pt[i].y;
 Sleep(20);
 }
 delete[] pt;
 DrawDibDraw(m_hDrawDib,dc.GetSafeHdc(),10,10,m_size.cx,m_size.cy,
```

```
&RGB32BITSBITMAPINFO,(LPVOID)m_clrSrc,
0,0,m_size.cx,m_size.cy,DDF_BACKGROUNDPA);
}
```

### 举一反三

根据本实例，读者可以：

- 实现图片的淡入淡出效果。

## 实例 116 图片百叶窗效果

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\03\116

### 实例说明

在一些多媒体教学软件中，常常可以看到图像的显示特效，包括图像之间的过渡特效，这些特效给软件本身增色不少。本实例实现将图片以百叶窗的形式显示出来，运行效果如图 3.18 所示。

### 技术要点

本实例主要通过在一个循环语句中使用 DrawDibDraw 函数来绘制图像，DrawDibDraw 函数和 BitBlt 函数作用相同，都是将一个设备环境中的数据复制到另一个设备环境中，DrawDibDraw 函数语法如下：

```
BOOL DrawDibDraw(HDRAWDIB hdd,HDC hdc,int xDst,int yDst,int dxDst,int dyDst,LPBITMAPINFOHEADER lpb,LPVOID lpBits,
int xSrc,int ySrc,int dxSrc,int dySrc,UINT wFlags);
```

参数说明：

- hdd: HDRAWDIB 对象。
- hdc: 设备环境句柄。
- xDst: 目标图像左顶点  $x$  轴坐标。
- yDst: 目标图像左顶点  $y$  轴坐标。
- dxDst: 目标图像的宽度。
- dyDst: 目标图像的高度。
- lpb: LPBITMAPINFOHEADER 结构对象。
- lpBits: 存储图像数据的缓存。
- xSrc: 源图像左顶点  $x$  轴坐标。
- ySrc: 源图像左顶点  $y$  轴坐标。
- dxSrc: 源图像的宽度。
- dySrc: 源图像的高度。
- wFlags: 表示绘画标记。

### 实现过程

- (1) 新建一个基于单文档的应用程序。
- (2) 向工程中加入图片资源，ID 属性设置为 IDB\_SHUTTER。在工程中添加对 vfw32.lib 库的引用。
- (3) 修改 ID 属性为 IDR\_MAINFRAME 菜单资源，在“查看”菜单下新建子菜单，设置 ID 属性为 ID\_VIEW，设置 Caption 属性为“百叶”，并添加实现函数 OnView。



图 3.18 图片百叶窗效果

(4) 在头文件 ShutterView.h 中添加文件的引用和变量声明:

```
#include "vfw.h"
HBITMAP m_bmpSrc;
COLORREF m_clrSrc;
CSize m_size;
HDRAWDIB m_hDrawDib;
```

(5) 在 CShutterView 方法中完成对 HDRAWDIB 对象的创建, 代码如下:

```
CShutterView::CShutterView()
{
 bstart=FALSE;
 m_hDrawDib=NULL;
 m_hDrawDib=DrawDibOpen();
 //加载图片资源
 HBITMAP hBmp=(HBITMAP)LoadImage(NULL,"bitmap.bmp",IMAGE_BITMAP,0,0,LR_LOADFROMFILE);
 HDC hMemDC=CreateCompatibleDC(NULL); //创建兼容的设备上下文
 if(hMemDC)
 {
 GetObject(hBmp,sizeof(bm),&bm);
 m_size=CSize(bm.bmWidth,bm.bmHeight); //获得位图的宽和高
 BITMAPINFOHEADER RGB32BITSBITMAPINFO=
 {sizeof(BITMAPINFOHEADER),bm.bmWidth,bm.bmHeight,1,32,BI_RGB,0,0,0,0,0};
 m_bmpSrc=CreateDIBSection(hMemDC,(BITMAPINFO*)&RGB32BITSBITMAPINFO,
 DIB_RGB_COLORS,(VOID**)&m_clrSrc,NULL,0);
 if(m_bmpSrc)
 {
 HBITMAP hOldBmp=(HBITMAP)SelectObject(hMemDC,m_bmpSrc);
 HDC hDC=CreateCompatibleDC(hMemDC);
 if(hDC)
 {
 HBITMAP hOldBmp2=(HBITMAP)SelectObject(hDC,hBmp);
 BitBlt(hMemDC,0,0,bm.bmWidth,bm.bmHeight,hDC,0,0,SRCCOPY); //绘制图像
 SelectObject(hDC,hOldBmp2);
 DeleteDC(hDC);
 }
 SelectObject(hMemDC,hOldBmp);
 }
 DeleteDC(hMemDC);
 }
}
```

(6) 开始显示百叶窗效果, 代码如下:

```
void CShutterView::OnView()
{
 bstart=TRUE;
 this->Invalidate();
}
```

(7) 在 OnDraw 中实现百叶窗效果, 代码如下:

```
void CShutterView::OnDraw(CDC* pDC)
{
 CShutterDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 if(bstart)
 {
 if(m_bmpSrc==NULL||m_hDrawDib==NULL)return;
 Invalidate();
 BOOL bDone=FALSE;
 int i=0,j;
 BITMAPINFOHEADER REG32BITSBITMAPINFO=
 {sizeof(BITMAPINFOHEADER),bm.bmWidth,bm.bmHeight,1,32,BI_RGB,0,0,0,0,0};
 DrawDibRealize(m_hDrawDib,pDC->GetSafeHdc(),true);
 for(i=0;i<20;i++)
 {
 for(j=i*j<m_size.cy;j+=20)
 DrawDibDraw(m_hDrawDib,pDC->GetSafeHdc(),j,0,1,
 m_size.cy,®32BITSBITMAPINFO,(LPVOID)m_clrSrc,
 j,0,1,m_size.cy,DDF_BACKGROUNDPALE);
 Sleep(200);
 }
 bstart=FALSE;
 }
}
```

## 举一反三

根据本实例，读者可以：

- 开发横向百叶窗。

## 实例 117 电影胶片特效

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\03\117

## 实例说明

本实例主要模仿电影播放时的效果实现图片从程序的一侧移动出来。选择“编辑”/“移动”菜单命令，图片就会从程序窗体的一侧移动出来，结果如图 3.19 所示。

## 技术要点

本实例主要应用了 DrawDib 库进行图像的绘制，有关该库的介绍可参见实例 115 图片马赛克效果。

## 实现过程

- 新建一个基于单文档的应用程序。
- 在工程中加入对 vfw32.lib 的应用。
- 在头文件中加入“#include "vfw.h"”。
- 添加菜单项，设置 ID 属性为 ID\_VIEW，Caption 属性为“移动”，并通过类向导添加菜单响应函数 OnView。
- 在 CFilmMoveViewView 类的头文件中声明变量：

```
class CFilmMoveViewView : public CView
{
protected:
 //加入变量声明
 HBITMAP m_bmpSrc;
 COLORREF* m_clrSrc;
 CSize m_size;
 HDRAWDIB m_hDrawDib;
 BITMAP bm;
}
```

- 在构造函数中初始化 HDRAWDIB 对象，代码如下：

```
CFilmMoveViewView::CFilmMoveViewView()
{
 m_hDrawDib=NULL;
 m_hDrawDib=DrawDibOpen();
 HBITMAP hBmp=
 (HBITMAP)LoadImage(NULL,"bitmap.bmp",IMAGE_BITMAP,0,0,LR_LOADFROMFILE);
 HDC hMemDC=CreateCompatibleDC(NULL);
 if(hMemDC)
 {
 GetObject(hBmp,sizeof(bm),&bm);
 m_size=CSize(bm.bmWidth,bm.bmHeight);
 BITMAPINFOHEADER RGB32BITSBITMAPINFO=
 {sizeof(BITMAPINFOHEADER),bm.bmWidth,bm.bmHeight,
 1,32,BI_RGB,0,0,0,0};
 m_bmpSrc=CreateDIBSection(hMemDC,(BITMAPINFO*)&RGB32BITSBITMAPINFO,
 DIB_RGB_COLORS,(VOID*)&m_clrSrc,NULL,0);
 if(m_bmpSrc)
 {
 HBITMAP hOldBmp=(HBITMAP)SelectObject(hMemDC,m_bmpSrc);
 HDC hDC=CreateCompatibleDC(hMemDC);
 if(hDC)
 {
 HBITMAP hOldBmp2=(HBITMAP)SelectObject(hDC,hBmp);
 BitBlt(hMemDC,0,0,bm.bmWidth,bm.bmHeight,hDC,
 0,0,SRCCOPY);
 }
 }
 }
}
```



图 3.19 电影胶片特效



```

 SelectObject(hDC,hOldBmp2);
 DeleteDC(hDC);
 }
 SelectObject(hMemDC,hOldBmp);
 DeleteDC(hMemDC);
}
}

```

(7) 在析构函数中实现 HDRAWDIB 对象的关闭和销毁, 代码如下:

```
CFilmMoveView::~CFilmMoveView()
```

```

{
 if(m_hDrawDib!=NULL)
 {
 DrawDibClose(m_hDrawDib);
 m_hDrawDib=NULL;
 }
 if(m_bmpSrc!=NULL)
 DeleteObject(m_bmpSrc);
}

```

(8) “查看” / “移动” 菜单的实现函数用于实现绘制图像, 代码如下:

```
void CFilmMoveView::OnView()
```

```

{
 if(m_bmpSrc==NULL||m_hDrawDib==NULL)return;
 Invalidate();
 BOOL bDone=false;
 int i=0;
 BITMAPINFOHEADER RGB32BITSBITMAPINFO=
 {sizeof(BITMAPINFOHEADER),m_size.cx,m_size.cy,1,32,BI_RGB,0,0,0,0,0};
 CPaintDC dc(this);
 while(!bDone)
 {
 if(i>m_size.cx)
 {
 i=m_size.cx;
 bDone=true;
 }
 DrawDibRealize(m_hDrawDib,dc.GetSafeHdc(),true);
 DrawDibDraw(m_hDrawDib,dc.GetSafeHdc(),10+m_size.cx-i,10,i,
 m_size.cy,&RGB32BITSBITMAPINFO,(LPVOID)m_clrSrc,
 0,0,i,m_size.cy,DDF_BACKGROUNDPALETTE);
 i+=10;
 Sleep(50);
 }
}

```

## 举一反三

根据本实例, 读者可以:

- 演示屏幕动画;
- 开发简单影片放映程序。

## 实例 118 翻转图片效果

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\03\118

## 实例说明

本实例实现图片的翻转效果, 选择“编辑” / “翻转”菜单命令, 图片就会翻转 90°。图像翻转前如图 3.20 所示, 图像翻转后如图 3.21 所示。

## 技术要点

本实例主要应用了 DrawDib 库进行图像的绘制, 有关该库的介绍可参见实例 115 图片马赛克效果。

## 实现过程

(1) 新建一个基于单文档的应用程序。



图 3.20 图像翻转前



图 3.21 图像翻转后

(2) 在工程中加入对 vfw32.lib 的引用。

(3) 在头文件加入 “#include “vfw.h””。

(4) 加入菜单项，设置 ID 属性为 ID\_VIEW，Caption 属性为“翻转”，通过类向导添加菜单响应函数 OnView。

(5) 在 CRotateViewView 类的头文件中声明变量：

```
class CRotateViewView : public CView
{
protected:
//加入变量声明
 HBITMAP m_bmpSrc;
 COLORREF *m_clrSrc;
 CSize m_sizeSrc;
 HBITMAP m_bmpDst;
 COLORREF *m_clrDst;
 CSize m_sizeDst;
 HDRAWDIB m_hDrawDib;
 int m_nType;
 BITMAP bm;
}
```

(6) 在构造函数中初始化 HDRAWDIB 对象，代码如下：

```
CRotateViewView::CRotateViewView()
{
 m_bmpSrc=NULL;
 m_bmpDst=NULL;
 m_nType=0;
 m_hDrawDib=DrawDibOpen();
 HBITMAP hBmp=(HBITMAP)LoadImage(NULL,"bitmap.bmp",IMAGE_BITMAP,0,0,
 LR_LOADFROMFILE);
 HDC hMemDC=CreateCompatibleDC(NULL);
 if(hMemDC)
 {
 GetObject(hBmp,sizeof(bm),&bm);
 m_sizeSrc=m_sizeDst=CSize(bm.bmWidth,bm.bmHeight);
 BITMAPINFOHEADER RGB32BITSBITMAPINFO=
 {sizeof(BITMAPINFOHEADER),bm.bmWidth,bm.bmHeight,1,32,
 BI_RGB,0,0,0,0,0};
 m_bmpSrc=CreateDIBSection(hMemDC,(BITMAPINFO*)&RGB32BITSBITMAPINFO,
 DIB_RGB_COLORS,(VOID*)&m_clrSrc,NULL,0);
 m_bmpDst=CreateDIBSection(hMemDC,(BITMAPINFO*)&RGB32BITSBITMAPINFO,
 DIB_RGB_COLORS,(VOID*)&m_clrDst,NULL,0);
 if(m_bmpSrc&& m_bmpDst)
 {
 HBITMAP hOldBmp=(HBITMAP)SelectObject(hMemDC,m_bmpSrc);
 HDC hDC=CreateCompatibleDC(hMemDC);
 if(hDC)
 {
 HBITMAP hOldBmp2=(HBITMAP)SelectObject(hDC,hBmp);
 BitBlt(hMemDC,0,0,bm.bmWidth,bm.bmHeight,hDC,0,0,
 SRCCOPY);
 SelectObject(hMemDC,m_bmpDst);
 BitBlt(hMemDC,0,0,bm.bmWidth,bm.bmHeight,hDC,0,0,
 SRCCOPY);
 SelectObject(hMemDC,hOldBmp2);
 }
 }
 }
}
```

```

DeleteObject(hDC);

}
SelectObject(hMemDC,hOldBmp);
}
DeleteObject(hMemDC);
DeleteObject(hBmp);
}
}

```

(7) 在析构函数中实现 HDRAWDIB 对象的关闭和销毁，代码如下：

```

CRotateViewView::~CRotateViewView()
{
 if(m_hDrawDib!=NULL)
 {
 DrawDibClose(m_hDrawDib);
 m_hDrawDib=NULL;
 }
 if(m_bmpDst!=NULL)
 DeleteObject(m_bmpDst);
 if(m_bmpSrc!=NULL)
 DeleteObject(m_bmpSrc);
}

```

(8) 在函数 OnDraw 中绘制图像，代码如下：

```

void CRotateViewView::OnDraw(CDC* pDC)
{
 CRotateViewDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 if(m_hDrawDib!=NULL&& m_bmpSrc!=NULL&& m_bmpDst!=NULL)
 {
 BITMAPINFOHEADER RGB32BITSBITMAPINFO=
 {sizeof(BITMAPINFOHEADER),m_sizeDst.cx,m_sizeDst.cy,
 1,32,BI_RGB,0,0,0,0,0};
 DrawDibRealize(m_hDrawDib,pDC->GetSafeHdc(),TRUE);
 DrawDibDraw(m_hDrawDib,pDC->GetSafeHdc(),0,0,m_sizeDst.cx,
 m_sizeDst.cy,&RGB32BITSBITMAPINFO,(LPVOID)m_clrDst,0,0,
 m_sizeDst.cx,m_sizeDst.cy,DDF_BACKGROUNDPALETTE);
 }
}

```

(9) “查看” / “翻转” 菜单的实现函数的代码如下：

```

void CRotateViewView::OnView()
{
 if(m_nType==1)return;
 if(m_bmpSrc!=NULL&&m_bmpDst!=NULL)
 {
 m_sizeDst.cx=m_sizeSrc.cy;
 m_sizeDst.cy=m_sizeSrc.cx;
 LONG x,y;
 for(y=0;y<m_sizeSrc.cy;y++)
 {
 for(x=0;x<m_sizeSrc.cx;x++)
 {
 m_clrDst[y+(m_sizeDst.cy-x-1)*m_sizeDst.cx]=
 m_clrSrc[x+y*m_sizeSrc.cx];
 }
 }
 m_nType=1;
 Invalidate();
 }
}

```

## 举一反三

根据本实例，读者可以：

- 开发图像的合成与处理的相关程序。

## 实例 119 图片浮雕效果

这是一个可以提高基础技能的实例

实例位置：光盘\mingr\soft\03\119

## 实例说明

在应用程序的启动界面或主界面当中，可以在窗体上绘制艺术图案以增强软件界面的美观性，

本实例演示的是将一幅图片绘制成浮雕图案。运行程序，单击“浮雕”按钮，效果如图 3.22 所示。

### 技术要点

浮雕效果实际上是将图片中每一点像素都进行了处理，首先循环遍历每一点的像素，分别取出像素的 R、G、B 元素值把这些值减去相邻像素的元素值再加上 128，因为这些元素值的取值在 0~255 之间，所以计算后超出了 255 则将元素值赋值为 255，小于 0 则赋值为 0。将这 3 个元素值重新组合赋予原来的像素。这样颜色就有了阶梯感。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个图片控件和两个按钮控件。
- (3) 主要程序代码如下：

```
void CRelievoDlg::RelievoImage()
{
 CDC* pDC = m_image.GetDC(); //获得设备上下文
 CRect m_rect;
 m_image.GetClientRect(m_rect); //获得控件的客户区域
 COLORREF color1, color2;
 BYTE r, g, b;
 for (int i = 0; i < m_rect.right; i++)
 for (int j = 0; j < m_rect.bottom; j++)
 {
 color1 = pDC->GetPixel(i, j); //获得每一点的颜色
 color2 = pDC->GetPixel(i+1, j+1);
 r = GetRValue(color1) - GetRValue(color2) + 128;
 g = GetGValue(color1) - GetGValue(color2) + 128;
 b = GetBValue(color1) - GetBValue(color2) + 128;
 //设置浮雕效果
 if (r > 255)
 r = 255;
 else if (r < 0)
 r = 0;
 if (g > 255)
 g = 255;
 else if (g < 0)
 g = 0;
 if (b > 255)
 b = 255;
 else if (b < 0)
 b = 0;
 pDC->SetPixel(i, j, RGB(r, g, b)); //设置颜色
 }
}
```

### 举一反三

根据本实例，读者可以：

- 绘制雕刻效果。

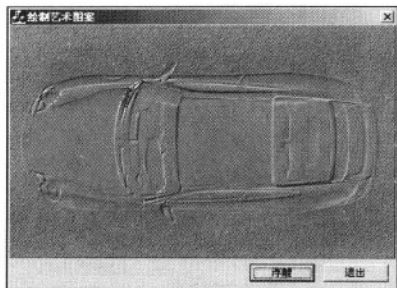


图 3.22 图片浮雕效果

## 3.4 图片颜色转换

### 实例 120 图像的锐化处理

本实例可以方便操作、提高效率

实例位置：光盘\mingrisoft\03\120

### 实例说明

图像的锐化处理能够使图像的主体轮廓更加清晰，因此在图像处理和图像识别程序中应用。



得非常广泛。本实例演示了图像的锐化效果,如图 3.23、图 3.24 所示。



图 3.23 锐化前的效果



图 3.24 锐化后的效果

## 技术要点

本实例中实现图像的锐化处理是通过调整像素点的颜色来实现的。在介绍“提取图片中的对象”实例时,讲解了CDC类的 GetPixel 方法和 SetPixel 方法可以获取和设置某一点的颜色,本实例通过这两个方法实现图像的锐化处理。

## 实现过程

(F) 新建一个基于对话框的应用程序。

(2) 在对话框中添加图片和按钮控件。

(3) 主要程序代码如下。

```
void CImageSharpDlg::OnSharp()
{
 CDC *m_dc;
 CRect m_rect;
 m_image1.GetClientRect(m_rect); //获得控件的客户区域
 m_dc = m_image1.GetDC(); //获得设备上下文
 int r1,g1,b1,r2,g2,b2;
 for (int i = 1; i < m_rect.right+1; i++) //根据宽度循环
 for (int j = 1; j < m_rect.bottom+1; j++) //根据高度循环
 {
 //锐化处理
 COLORREF color = m_dc->GetPixel(i,j);
 COLORREF nextcolor = m_dc->GetPixel(i-1,j-1);
 r1 = (color & 0xFF);
 g1 = (int)(color & 62580) / 256;
 b1 = (int)(color & 0xFF0000) / 65536;
 r2 = (nextcolor & 0xFF);
 g2 = (int)(nextcolor & 62580) / 256;
 b2 = (int)(nextcolor & 0xFF0000) / 65536;
 r1 += (r1 - r2) / 2;
 g1 += (g1 - g2) / 2;
 b1 += (b1 - b2) / 2;
 if (r1 > 255)
 r1 = 255;
 if (r1 < 0)
 r1 = 0;
 if (b1 > 255)
 b1 = 255;
 if (b1 < 0)
 b1 = 0;
 if (g1 > 255)
 g1 = 255;
 if (g1 < 0)
 g1 = 0;
 m_dc->SetPixel(i,j,RGB(r1,g1,b1));
 }
}
```

## 举一反三

根据本实例,读者可以:

● 设计渐变效果程序启动界面。

## 实例 121 图片反色处理

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\03\121

### 实例说明

图片反色处理是将图片中的像素值取反。例如，原来的白色像素点取反后会成为黑色的像素点。图片的反色处理是一个逆运算过程，即对一幅图片进行两次取反处理，还应是原来的图片效果。本实例实现了图片的反色处理。反色处理前后的效果如图 3.25、图 3.26 所示。



图 3.25 处理前的效果图



图 3.26 处理后的效果图

### 技术要点

要实现图像的反色处理可以有多种方法。例如可以获取图片中每个像素点的颜色值，然后对颜色值进行取反，代码如下：

```
CDC* pDC = m_image.GetDC();
CRect m_rect;
m_image.GetClientRect(m_rect);

BYTE r,g,b;
for (int i=1; i<=m_rect.Width();i++)
 for (int j=1;j<=m_rect.Height();j++)
 {
 COLORREF clr= pDC->GetPixel(i,j);
 r = GetRValue(clr);
 g = GetGValue(clr);
 b = GetBValue(clr);

 r = abs(255-r);
 g = abs(255-g);
 b = abs(255-b);
 pDC->SetPixel(i,j,RGB(r,g,b));
 }
```

此外，还有一种更简单的方法，就是调用 CDC 类的 InvertRgn 方法，该方法将指定区域的颜色取反，语法如下：

```
BOOL InvertRgn(CRgn* pRgn);
```

参数说明：

- pRgn：是一个区域对象指针。

与第一种方法相比，该方法计算速度更快，而且也相对简单，本实例也是采用该方法实现的，关键代码如下：

```
CDC* pDC = m_image.GetDC();
CRect m_rect;
m_image.GetClientRect(m_rect);
CRgn m_rgn;
m_rgn.CreateRectRgn(m_rect.left,m_rect.top,m_rect.right,m_rect.bottom);
pDC->InvertRgn(&m_rgn);
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加图片和按钮控件。
- (3) 主要程序代码如下：

```
void CReverseColorDlg::OnOK()
{
 CDC* pDC = m_image.GetDC(); //获得设备上下文
 CRect m_rect;
 m_image.GetClientRect(m_rect); //获得控件客户区域
 CRgn m_rgn;
 m_rgn.CreateRectRgn(m_rect.left,m_rect.top,m_rect.right,m_rect.bottom); //设置区域
 pDC->InvertRgn(&m_rgn); //颜色取反处理
}
```

## 举一反三

根据本实例，读者可以：

- 设计具有透明效果的图像。

## 实例 122 图像的灰度化转换

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\03\122

## 实例说明

在图像识别领域，经常涉及图像的灰度化转换，即将彩色图像转换为黑白图像。因为黑白图像更容易进行运算。本实例将实现图像的灰度化转换，效果如图 3.27、图 3.28 所示。



图 3.27 转换前图像

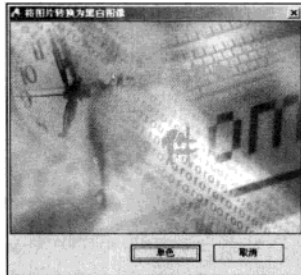


图 3.28 转换后图像

## 技术要点

实现图像的灰度化转换没有一定的标准，通常根据图片中像素的 RGB 分量以及它们的权重来获取。本实例中 RGB 分量的权重分别为 0.38、0.49、0.1。在其他应用中，用户可以根据实际情况设置不同的权重。在图像的灰度化过程中，首先需要获取像素点的红、绿、蓝分量值，然后将其乘以相应的分量，最后重新设置像素的颜色，例如下面的代码：

```
m_color = pDC->GetPixel(i,j);
r = GetRValue(m_color);
g = GetGValue(m_color);
b = GetBValue(m_color);
m_gray = (0.38*r+0.49*g+0.1*b);
m_color = RGB(m_gray,m_gray,m_gray);
pDC->SetPixel(i,j,m_color);
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。

(2) 在对话框中添加图片控件和按钮控件。

(3) 主要程序代码如下:

```
void CSingleImageDlg::OnOK()
{
 CDC* pDC = m_image.GetDC(); //获得设备上下文
 CRect m_rect;
 m_image.GetClientRect(m_rect); //获得控件区域
 COLORREF m_color;
 DWORD m_gray;
 BYTE r,g,b;
 for (int i = 0; i<m_rect.right;i++)
 for (int j = 0;j<m_rect.bottom;j++)
 {
 m_color = pDC->GetPixel(i,j); //获得颜色
 r = GetRValue(m_color);
 g = GetGValue(m_color);
 b = GetBValue(m_color);
 m_gray = (0.38*r+0.49*g+0.1*b); //设置灰度颜色值
 m_color = RGB(m_gray,m_gray,m_gray);
 pDC->SetPixel(i,j,m_color); //用灰度颜色画点
 }
}
```

### 举一反三

根据本实例,读者可以:

- 利用抠图技术设计特殊形状的窗体。

## 实例 123 显示 JPG 图片

这是一个可以启发思维的实例

实例位置: 光盘\mingrisoft\03\123

### 实例说明

用过 Delphi 的读者知道,使用 VCL 提供的 TImage 可以方便地显示 JPG 图像。在 Visual C++ 中, MFC 类库却没有提供相应的控件显示 JPG 图像。本实例实现了 JPG 图片的显示,效果如图 3.29 所示。

### 技术要点

JPG 图像采用压缩形式存储,存储结构比较复杂。为了简单化,可以采用流的形式读取 JPG 文件。首先利用 CFile 对象读取 JPG 文件,获取 JPG 文件的长度,然后利用 GlobalAlloc 函数在堆中划分一个和文件大小相同的区域,接着调用 GlobalLock 函数锁定该区域,返回一个指向堆的指针,将文件数据写入该区域,最后调用 CreateStreamOnHGlobal 函数在堆中创建流对象,调用 OleLoadPicture 函数根据流对象创建图像对象 IPicture;调用 IPicture 的 Render 方法将图像显示在指定的画布上。

在实现 JPG 图像的过程中,需要用到多个函数,下面逐一进行介绍。

(1) GlobalAlloc 函数。该函数用于在堆上分配指定大小的空间。语法如下:

```
HGLOBAL GlobalAlloc(UINT uFlags, DWORD dwBytes);
```

参数说明:

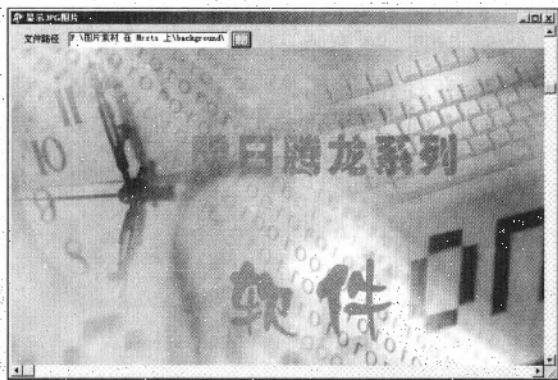


图 3.29 显示 JPG 图片



- **uFlags**: 标识如何分配空间, 为 **GMEM\_FIXED** 时表示空间是固定的, 为 **GMEM\_MOVEABLE** 时表示空间可以移动。
- **dwBytes**: 标识分配空间的大小。
- **返回值**: 分配的空间对象句柄。

(2) **GlobalLock** 函数。该函数用于锁定一个全局内存对象, 并返回一个指向内存的指针。

语法如下:

```
LPVOID GlobalLock(HGLOBAL hMem);
```

参数说明:

- **hMem**: 全局内存对象句柄, 通常是 **GlobalAlloc** 函数的返回值。
- **返回值**: 函数返回指向全局内存的指针。

(3) **CreateStreamOnHGlobal** 函数。该函数用于创建一个存储在堆中的流对象。语法如下:

```
WINOLEAPI CreateStreamOnHGlobal(HGLOBAL hGlobal, BOOL fDeleteOnRelease, LPSTREAM * ppstm);
```

参数说明:

- **hGlobal**: 标识全局内存对象句柄。
- **fDeleteOnRelease**: 确定在流对象释放时是否释放全局内存空间。
- **ppstm**: 标识 **IStream** 接口指针。

(4) **OleLoadPicture** 函数。该函数根据流对象的内容创建一个图像对象。语法如下:

```
STDAPI OleLoadPicture(IStream * pStream, LONG lSize, BOOL fRunmode, REFIID riid, VOID ppvObj);
```

参数说明:

- **pStream**: 标识流对象指针。
- **lSize**: 标识从流中读取数据的数量。
- **fRunmode**: 确定是否采用与图像初始属性相反的属性值。
- **riid**: 确定接口指针的类型。
- **ppvObj**: 标识与 **riid** 对应的图像对象指针。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框中添加静态文本、编辑框、按钮和图片控件。

(3) 处理“...”按钮的单击事件, 加载并显示 JPG 图像, 代码如下:

```
void CShowJPGDlg::OnOk()
```

```
{
 CFileDialog m_dlg(TRUE, "JPG", NULL, NULL, "JPG(*.jpg)|*.JPG|gif*.gif", this);
 if (m_dlg.DoModal() != IDOK)
 {
 CString m_filename = m_dlg.GetPathName();
 m_dir.SetWindowText(m_filename);
 CFile m_file(m_filename, CFile::modeRead);
 // 获取文件长度
 DWORD m_filelen = m_file.GetLength();
 // 在堆上分配空间
 HGLOBAL m_hglobal = GlobalAlloc(GMEM_MOVEABLE, m_filelen);
 LPVOID pvdata = NULL;
 // 锁定堆空间, 获取指向堆空间的指针
 pvdata = GlobalLock(m_hglobal);
 // 将文件数据读取到堆中
 m_file.ReadHuge(pvdata, m_filelen);
 IStream* m_stream;
 GlobalUnlock(m_hglobal);
 // 在堆中创建流对象
 CreateStreamOnHGlobal(m_hglobal, TRUE, &m_stream);
 // 利用流加载图像
 OleLoadPicture(m_stream, m_filelen, TRUE, IID_IPicture, (LPVOID*)&m_picture);
 m_picture->get_Width(&m_width);
 m_picture->get_Height(&m_height);
 }
}
```

```

CDC* dc = GetDC();
m_IsShow = TRUE;
CRect rect;
GetClientRect(rect);
SetScrollRange(SB_VERT,0,(int)(m_height/26.45)-rect.Height());
SetScrollRange(SB_HORZ,0,(int)(m_width/26.45)-rect.Width());
m_picture->Render(*dc,1,50,(int)(m_width/26.45),(int)(m_height/26.45),
0,m_height,m_width,-m_height,NULL);
}
}

```

### 举一反三

根据本实例，读者可以：

- 浏览 GIF 动画。

## 3.5 图形转换与缩放

### 实例 124 将位图转换为 JPG

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\03\124

#### 实例说明

在进行图像处理时，经常涉及图形转换。本实例实现了将位图转换为 JPG 格式图像，效果如图 3.30 所示。

#### 技术要点

JPG 文件采用了复杂的数据压缩技术。本实例通过 OCX 控件 CJPG 实现从位图到 JPG 的转换。有关 CJPG 的源代码可以在本实例源代码目录下的“JPG”文件夹下找到。

#### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加图片控件和按钮控件。
- (3) 注册 JPGXControl.ocx 控件，导入 OCX 控件 CJPG。

(4) 主要程序代码如下：

```

void CRevertPictureDlg::OnOK()
{
 m_JPG.SetBmpFile(m_filename); //设置位图文件路径
 m_JPG.SetQuality((long)90);
 CFileDialog m_dlg(FALSE,"JPG",NULL,NULL,"JPG图像(JPG)*.JPG",this); //构造另存为对话框
 if (m_dlg.DoModal() != IDOK)
 {
 m_JPG.BmpToJPG(m_dlg.GetPathName()); //转换图片
 }
}

```

### 举一反三

根据本实例，读者可以：

- 将 JPG 转换为位图。



图 3.30 将位图转换为 JPG

## 实例 125 将位图转为 GIF 图标

本实例是一个提高效率的程序

实例位置: 光盘\mingrisoft\03\125

## 实例说明

由于位图文件通常很大,因此网上的许多动画和图片都是采用 GIF 或 JPG 格式显示。本实例将实现位图转换为 GIF 格式,效果如图 3.31 所示。

## 技术要点

在 Visual C++ 中没有提供相应的类或函数将位图转换为 GIF,本实例使用了 OCX 控件 CGIF。通过调用 CGIF 控件的 SaveToFile 方法实现位图的转换。CGIF 控件的源代码位于本实例源代码下的 GIF 文件夹下。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加图片控件、按钮和编辑框控件。
- (3) 注册 GIFXControl.ocx 控件,导入 OCX 控件 CGIF。

(4) 主要程序代码如下:

```
void CConvertGifDlg::OnConvert()
{
 if (!m_FileName.IsEmpty())
 {
 m_Gif.SetBmpFile(m_FileName); //设置待转换的文件路径
 CFileDialog dlg(FALSE, "gif", NULL, NULL, "GIF 文件(GIF)*.gif", this); //构造另存为对话框
 if (dlg.DoModal() == IDOK)
 {
 m_Gif.SaveToFile(dlg.GetPathName()); //转换为GIF图标
 }
 }
}
```



图 3.31 将位图转为 GIF 图标

## 举一反三

根据本实例,读者可以:

- 将 GIF 转换为位图。

## 实例 126 图片的平滑缩放

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\03\126

## 实例说明

在浏览图片的时候,如果被浏览的图片失真或者大于浏览区的界面,可以将图片缩小或放大后再进行浏览。本实例实现了等比例平滑缩放图片的功能。运行程序,单击“打开”按钮,打开要进行平滑缩放的图片文件,将打开的文件显示在窗体中。通过拖动窗体中间部分的滑块控件设置图片缩放的比例,单击“缩放”按钮,完成图片的平滑缩放,如图 3.32 所示。



图 3.32 图片的平滑缩放


## 技术要点

本实例通过滑块控件设置图片的缩放比率，然后使用 `StretchBlt` 函数将图片按比例进行缩放，最后通过带有滚动条的无模式对话框配合图片控件显示缩放后的图片。利用 `SetRange` 方法设置滑块控件两侧的刻度，语法如下：

```
void SetRange(int nMin, int nMax, BOOL bRedraw = FALSE);
```

参数说明：

- `nMin`：滑块控件的最小值。
- `nMax`：滑块控件的最大值。
- `bRedraw`：指定滑块控件是否重画以反映滑标的变化。如果这个参数为 `TRUE`，滑标将被重画，否则不被重画。

 注意：`StretchBlt` 函数的使用方法可以参照实例 1 E3 放大和缩小图片。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 创建一个对话框，并选择 `Horizontal Scroll` 和 `Vertical Scroll` 风格。为对话框添加类 `CFrameDlg`，在该类中设置滚动条的滑块位置等参数。
- (3) 向窗体中添加一个滑块控件、两个图片控件和两个按钮控件。
- (4) 主要程序代码如下：

```
void CSmoothnessDlg::OnButopen()
{
 CFileDialog m_dialog(true, "bmp", NULL, OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
 "位图文件(*.bmp)|*.bmp", this); //构造打开对话框
 if (m_dialog.DoModal() != IDOK)
 {
 CString str;
 str = m_dialog.GetPathName();
 m_hBitmap = (HBITMAP)::LoadImage(AfxGetInstanceHandle(), str, IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE | LR_DEFAULTCOLOR | LR_DEFAULTSIZE); //加载位图资源
 if (m_hBitmap != NULL)
 {
 m_picture.SetBitmap(m_hBitmap); //显示图片
 }
 BITMAP bmp;
 GetObject(m_hBitmap, sizeof(bmp), &bmp); //获得图片信息
 CRect rect;
 dlg.GetClientRect(rect);
 int xpos = dlg.GetScrollPos(SB_HORZ);
 if (xpos != 0)
 dlg.ScrollWindow(xpos, 0); //恢复窗口的水平滚动区域
 int ypos = dlg.GetScrollPos(SB_VERT);
 if (ypos != 0)
 dlg.ScrollWindow(0, ypos); //恢复窗口的垂直滚动区域
 SCROLLINFO vinfo;
 vinfo.cbSize = sizeof(vinfo);
 vinfo.fMask = SIF_ALL;
 vinfo.nPage = bmp.bmHeight/10;
 vinfo.nMax = bmp.bmHeight-rect.Height()+bmp.bmHeight/10;
 vinfo.nMin = 0;
 vinfo.nTrackPos = 0;
 vinfo.nPos = 0;
 //设置垂直滚动条信息
 dlg.SetScrollInfo(SB_VERT, &vinfo);
 vinfo.fMask = SIF_ALL;
 vinfo.nPage = bmp.bmWidth/10;
 vinfo.nMax = bmp.bmWidth-rect.Width()+bmp.bmWidth/10;
 vinfo.nMin = 0;
 vinfo.nPos = 0;
 vinfo.nTrackPos = 0;
 vinfo.cbSize = sizeof(vinfo);
 //设置水平滚动条信息
 dlg.SetScrollInfo(SB_HORZ, &vinfo);
 }
}
```



```

void CSmoothnessDlg::OnReleasedcaptureSlider1(NMHDR* pNMHDR, LRESULT* pResult)
{
 CString str;
 str.Format("%d%%", m_slider.GetPos()); //将滑块当前位置格式化为字符串
 m_edit.SetWindowText(str); //显示当前位置
 *pResult = 0;
}

void CSmoothnessDlg::OnButdraw()
{
 CRect rect;
 dlg.GetClientRect(rect);
 int xpos = dlg.GetScrollPos(SB_HORZ);
 if (xpos != 0)
 dlg.ScrollWindow(xpos, 0); //恢复窗口的水平滚动区域
 int ypos = dlg.GetScrollPos(SB_VERT);
 if (ypos != 0)
 dlg.ScrollWindow(0, ypos); //恢复窗口的垂直滚动区域
 CDC* pDC = m_picture.GetDC();
 //将位图选进设备场景中
 CDC memdc;
 memdc.CreateCompatibleDC(pDC);
 memdc.SelectObject(m_hBitmap);
 BITMAP bmp;
 GetObject(m_hBitmap, sizeof(bmp), &bmp); //获得图片信息
 int x, y;
 x = bmp.bmWidth * m_slider.GetPos() / 100;
 y = bmp.bmHeight * m_slider.GetPos() / 100;
 m_picture.MoveWindow(rect.left, rect.top, x, y, true);
 pDC->StretchBlt(rect.left, rect.top, x, y, memdc, 0, 0, bmp.bmWidth, bmp.bmHeight, SRCCOPY); //绘制图片
 memdc.DeleteDC();
 SCROLLINFO vinfo;
 vinfo.cbSize = sizeof(vinfo);
 vinfo.fMask = SIF_ALL;
 vinfo.nPage = y / 10;
 vinfo.nMax = y - rect.Height() + y / 10;
 vinfo.nMin = 0;
 vinfo.nTrackPos = 0;
 vinfo.nPos = 0;
 //设置垂直滚动条信息
 dlg.SetScrollInfo(SB_VERT, &vinfo);
 vinfo.fMask = SIF_ALL;
 vinfo.nPage = x / 10;
 vinfo.nMax = x - rect.Width() + x / 10;
 vinfo.nMin = 0;
 vinfo.nPos = 0;
 vinfo.nTrackPos = 0;
 vinfo.cbSize = sizeof(vinfo);
 //设置水平滚动条信息
 dlg.SetScrollInfo(SB_HORZ, &vinfo);
}

```

### 举一反三

根据本实例，读者可以：

- 开发图像编辑软件。

## 3.6 图像的剪切、合成与识别

### 实例 127 图像的剪切

这是一个可以启发思维的实例

实例位置：光盘\mingr\soft\03\127

#### 实例说明

几乎所有的图像编辑软件都提供了图像的剪切功能，用户可以根据需要剪切图像的某一部分进行处理，本实例也实现了这一功能，运行程序，单击“剪切”按钮，将截取源图像的某一区域到目标图像中，如图 3.33 所示。

## 技术要点

实现图像的剪切可以使用 CRgn 类创建一个剪切的区域, 然后利用目标设备上下文 CDC 选中该剪切的区域, 在该区域绘制图像, 最后在源设备上下文中将剪切的区域设置为白色的背景。CRgn 封装了 Windows 图像设备接口的区域对象, 该区域可以是椭圆形或多边形。用户可以通过该类为 CDC 定义一个剪切的区域。CRgn 类的主要方法如下:

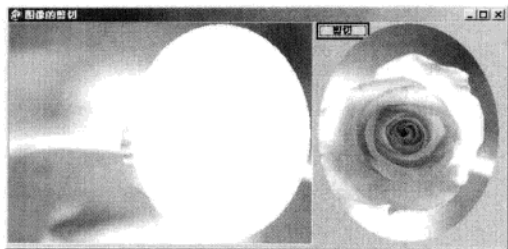


图 3.33 图像的剪切

(1) CreateRectRgn 方法。该方法用于创建一个矩形区域。语法如下:

```
BOOL CreateRectRgn(int x1, int y1, int x2, int y2);
```

参数说明:

- x1、y1: 标识矩形区域的左上角坐标。
- x2、y2: 标识矩形区域的右下角坐标。

(2) CreateEllipticRgn 方法。该方法用于创建一个椭圆形的区域。语法如下:

```
BOOL CreateEllipticRgn(int x1, int y1, int x2, int y2);
```

参数说明:

- x1、y1: 标识椭圆的外接矩形的左上角坐标。
- x2、y2: 标识椭圆的外接矩形的右下角坐标。

(3) CreatePolygonRgn 方法。该方法用于创建一个多边形的区域。语法如下:

```
BOOL CreatePolygonRgn(LPPOINT lpPoints, int nCount, int nMode);
```

参数说明:

- lpPoints: 是 CPoint 类型数组指针, 用于标识多边形的顶点坐标。
- nCount: 标识 lpPoints 数组中元素的数量。
- nMode: 标识区域的填充模式。

(4) CombineRgn 方法。该方法用于组合两个区域。语法如下:

```
int CombineRgn(CRgn* pRgn1, CRgn* pRgn2, int nCombineMode);
```

参数说明:

- pRgn1、pRgn2: 标识组合区域的两个指针。
- nCombineMode: 确定区域的组合模式, 可选值如下。
  - RGN\_AND: 使用两个区域的重叠部分。
  - RGN\_COPY: 使用第一个区域。
  - RGN\_DIFF: 创建由第一区域组成的区域, 而不包含第二个区域。
  - RGN\_OR: 使用两个区域的共同区域。
  - RGN\_XOR: 使用两个区域的非重叠部分。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加图片控件和按钮控件。
- (3) 主要程序代码如下:

```
void CCutImageDlg::OnOK()
{
 CBitmap m_bitmap;
 HBITMAP m_hbitmap = m_sourceimage.GetBitmap();
 //附加位图句柄
 m_bitmap.Attach(m_hbitmap);
```

```

CDC* m_dc = m_cutimage.GetDC();
CRgn m_rgn;
BITMAP m_bitinfo;
m_bitmap.GetBitmap(&m_bitinfo);
//创建一个剪切区域
m_rgn.CreateEllipticRgn(150,1,m_bitinfo.bmWidth-1,m_bitinfo.bmHeight-1);
CDC* m_sourcecdc = m_sourceimage.GetDC();
//选中剪切区域
m_dc->SelectClipRgn(&m_rgn,RGN_COPY);
m_dc->BitBlt(0,0,m_bitinfo.bmWidth,m_bitinfo.bmHeight,m_sourcecdc,0,0,SRCCOPY); //绘制剪切图像
m_sourcecdc->SelectClipRgn(&m_rgn,RGN_COPY);
m_sourcecdc->BitBlt(0,0,m_bitinfo.bmWidth,m_bitinfo.bmHeight,m_dc,0,0,WHITENESS);//绘制源图像
m_bitmap.Detach();
}

```

## 举一反三

根据本实例，读者可以：

- 开发具有合成图层功能的图像处理软件。

## 实例 128 图像的合成

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\03\128

## 实例说明

在许多图像处理软件中，都提供了图像的合成功能。例如在 PhotoShop 中，用户可以利用图层技术合成图像。本实例实现了图像的合成技术，效果如图 3.34 所示。

## 技术要点

本实例中实现图像合成是采用设备上下文 CDC 类的 BitBlt 方法实现的。CDC 类提供了多个绘制图像的方法，常用的主要有 BitBlt 和 StretchBlt 两个方法，下面分别介绍这两个方法。



图 3.34 图像的合成

(1) BitBlt 方法。该方法将位图从源设备区域复制到目标设备区域。语法如下：

```
BOOL BitBlt(int x, int y, int nWidth, int nHeight, CDC* pSrcDC, int xSrc, int ySrc, DWORD dwRop);
```

参数说明：

- x、y：标识目标区域的左上角坐标。
- nWidth、nHeight：确定复制位图的宽度和高度。
- pSrcDC：标识源设备上下文指针。
- xSrc、ySrc：标识源位图的左上角坐标。
- dwRop：确定复制模式，通常为 SRCCOPY。

(2) StretchBlt 方法。该方法将位图从源设备区域复制到目标设备区域。与 BitBlt 方法不同的是，StretchBlt 方法在必要的时候会延长或压缩位图区域以适合目标区域。语法如下：

```
BOOL StretchBlt(int x, int y, int nWidth, int nHeight, CDC* pSrcDC, int xSrc, int ySrc, int nSrcWidth, int nSrcHeight, DWORD dwRop);
```

参数说明：

- x、y：标识目标区域的左上角坐标。
- nWidth、nHeight：确定复制位图的宽度和高度。
- pSrcDC：标识源设备上下文指针。
- xSrc、ySrc：标识源位图的左上角坐标。
- nSrcWidth、nSrcHeight：标识源位图的宽度和高度。

- dwRop: 确定复制模式, 通常为 SRCCOPY。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加 Button 和 Picture 控件。
- (3) 主要程序代码如下:

```
void CCombineImageDlg::OnOK()
{
 //获取背景图片设备上下文
 CDC* m_grounddc = m_back.GetDC();
 //获取子图片设备上下文
 CDC* m_babydc = m_baby.GetDC();
 CBitmap m_bitmap; //位图对象
 BITMAP m_bitinfo; //位图信息
 int m_height, m_width;
 m_bitmap.Detach();
 m_bitmap.Attach((HBITMAP)m_baby.GetBitmap());
 //获取位图大小
 m_bitmap.GetObject(sizeof(m_bitinfo), &m_bitinfo);
 m_width = m_bitinfo.bmWidth; //图像宽度
 m_height = m_bitinfo.bmHeight; //图像高度
 //在背景图片的指定区域绘制图像
 m_grounddc->BitBlt(130, 100, m_width, m_height, m_babydc, 0, 0, SRCCOPY);
 //将句柄与位图对象分离
 m_bitmap.Detach();
}
```

## 举一反三

根据本实例, 读者可以:

- 实现图像的剪切。

## 实例 129

## 获取鼠标任意位置的颜色值

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\03\129

## 实例说明

在 Windows 的画图程序中, 打开“编辑颜色”窗口时, 当鼠标在颜色区域移动时, 右方的显示区域和下方的编辑框中会显示相应的颜色和颜色值。本实例实现了该功能, 效果如图 3.35 所示。

## 技术要点

获取某一点的颜色很容易, 只要得到当前鼠标下的设备上下文 CDC 类就可以了, 因为调用 CDC 类的 GetPixel 方法可以获取某一点的颜色值。但是, 通过颜色值如何获得红、绿、蓝三原色的值呢? Visual C++ 提供了 3 个宏, 用于获取某一颜色的红、绿、蓝三原色, 这 3 个宏分别如下。

- (1) GetRValue 宏。该宏用于获取指定颜色的红颜色值。

语法如下:

```
BYTE GetRValue(DWORD rgb);
```

参数说明:

- rgb: 标识一个颜色值。
- 返回值: 指定颜色的红色值。

- (2) GetGValue 宏。该宏用于获取指定颜色的绿颜色值。语法如下:

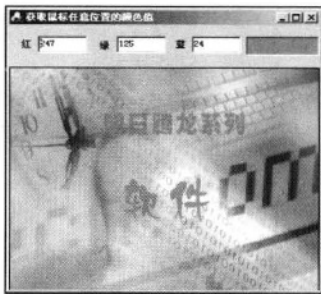


图 3.35 获取鼠标任意位置的颜色值



BYTE GetGValue(DWORD rgb);

参数说明:

- rgb: 标识一个颜色值。
- 返回值: 指定颜色的绿色值。

(3) GetBValue 宏。该宏用于获取指定颜色的蓝色值。语法如下:

BYTE GetBValue(DWORD rgb);

参数说明:

- rgb: 标识一个颜色值。
- 返回值: 指定颜色的蓝色值。

在 MFC 应用程序中, 颜色值通常采用 COLORREF 类型, 在设置颜色值时, 可以利用 RGB 宏将红、绿、蓝三原色组合为一个 COLORREF 类型颜色值。例如获取红颜色, 代码如下:

RGB(255,0,0); //获取红颜色

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加静态文本控件、图片控件和编辑框控件。
- (3) 主要程序代码如下:

```
void CFetchColorDlg::OnMouseMove(UINT nFlags, CPoint point)
```

```
{
 CDialog::OnMouseMove(nFlags, point);
 CWnd* temp = ChildWindowFromPoint(point);
 if (temp)
 {
 CDC* m_dc = temp->GetDC();
 COLORREF m_color;
 this->MapWindowPoints(temp, &point, 1);
 //获取颜色值
 m_color = m_dc->GetPixel(point);
 int r, g, b;
 r = GetRValue(m_color);
 g = GetGValue(m_color);
 b = GetBValue(m_color);
 //在编辑框中显示颜色值
 CString str;
 str.Format("%i", r);
 m_red.SetWindowText(str);
 str.Format("%i", g);
 m_green.SetWindowText(str);
 str.Format("%i", b);
 m_blue.SetWindowText(str);
 //使用当前颜色填充区域
 CRect m_rect;
 m_test.GetClientRect(m_rect);
 CDC* dc = m_test.GetDC();
 CBrush m_brush(RGB(r, g, b));
 dc->FillRect(m_rect, &m_brush);
 }
}
```

## 举一反三

根据本实例, 读者可以:

- 设计颜色拾取器。

## 实例 130 提取图片中的对象

这是一个可以提高分析能力的实例

实例位置: 光盘\mingrisoft\03\130

## 实例说明

在图像识别领域中, 图像提取是最基础也是最重要的一个环节。如果前期图像提取错误,

无论识别技术多么高超,最终也不会识别出正确的对象。  
本实例将实现从图像中提取人物轮廓,效果如图3.36所示。

### 技术要点

要从图像中提取某个对象,首先需要观察提取对象的特征。例如本实例中提取的对象,轮廓是由黑色的像素构成,因此只要将黑色像素提取出来,则对象的轮廓也就描述出来了。设备上下文 CDC 类提供了 GetPixel 方法和 SetPixel 方法,用于获取或设置某个点的颜色。下面详细介绍这两个方法。

(1) GetPixel 方法。该方法用于获取某一点的颜色值。

语法如下:

```
COLORREF GetPixel(int x, int y) const;
COLORREF GetPixel(POINT point) const;
```

参数说明:

- x、y、point: 标识坐标点。
- 返回值: 坐标点的颜色值。

(2) SetPixel 方法。该方法用于设置某一点的颜色值。语法如下:

```
COLORREF SetPixel(int x, int y, COLORREF crColor);
COLORREF SetPixel(POINT point, COLORREF crColor);
```

参数说明:

- x、y、point: 标识坐标点。
- crColor: 标识设置的颜色值。
- 返回值: 坐标点实际显示的颜色值。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加图片控件和按钮控件。
- (3) 主要程序代码如下:

```
void CFetchObjectDlg::OnFetch()
{
 CDC* dc = m_image.GetDC();
 CDC* m_dc = m_demo.GetDC();
 CRect m_rect;
 m_image.GetClientRect(&m_rect); //获得控件的客户区域
 int x,y;
 for (x=0;x<m_rect.right;x++) //根据宽度循环
 for (y=0;y<m_rect.bottom;y++) //根据高度循环
 {
 COLORREF m_color;
 m_color = dc->GetPixel(x,y); //获得当前颜色
 //判断是否为要获取的颜色
 if ((m_color==RGB(255,255,255))||(m_color==RGB(0,0,0))||(m_color==RGB(252,197,30)))
 {
 m_dc->SetPixel(x,y,m_color);
 }
 }
}
```

### 举一反三

根据本实例,读者可以:

- 开发简易的图像识别程序。



图 3.36 提取图片中的对象

## 实例 131 手写数字识别

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\03\131

## 实例说明

如今许多软件都提供了文字识别功能。例如在一些手机的应用软件中，用户能够通过手写文字实现短信发送，其中就涉及了文字的识别技术。本实例实现了手写数字的识别，采用联机字符识别技术，效果如图 3.37 所示。

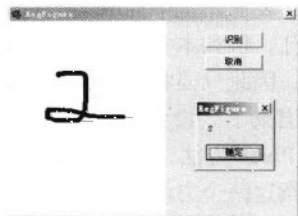


图 3.37 手写数字识别

## 技术要点

手写数字识别的难度在于其形状极多。对于规范的手写数字，可以采用模板匹配的方法。

但是，由于每个人的字体不尽相同，导致数字或大或小、或胖或瘦，如图 3.38 所示，采用模板匹配就行不通了。



图 3.38 手写数字

在图 3.38 中，虽然“2”的写法不同，但人眼一下就能识别出它们是“2”。分析原因，这两个字都是向右、向左下、向右的书写顺序。它们的特征在于笔顺相同。本实例就是以数字的笔顺为特征区别手写数字的。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加图片控件和按钮控件。
- (3) 主要程序代码如下：

```
void CRegFigureDlg::OnReg()
{
 m_curpen = 0;
 if (m_Figure.Direction[0] == down) //判断1
 if (m_Figure.Direction[1] == none) //只有一笔记录
 {
 if (m_Figure.DotCount == 1)
 {
 MessageBox("1");
 for (int i = 0; i < 16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i] = none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
 }
 }
 if (m_Figure.Direction[0] == right) //判断数字7
 if (m_Figure.Direction[1] == down)
 if (m_Figure.Direction[2] == none)
 {
 if (m_Figure.DotCount == 1)
 {
 MessageBox("7");
 for (int i = 0; i < 16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i] = none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
 }
 }
 if (m_Figure.Direction[0] == right) //判断数字2
```



```

if (m_Figure.Direction[1]==down)
{
 if (m_Figure.DotCount == 1)
 {
 if (m_Figure.Direction[2]==left)
 {
 if (m_Figure.Direction[3]==right)
 if (m_Figure.Direction[4]==none)
 {
 MessageBox("2");
 for (int i =0; i<16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i]= none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
 }
 }
 else if (m_Figure.Direction[2]==right)
 {
 if (m_Figure.Direction[3]==none)
 {
 MessageBox("2");
 for (int i =0; i<16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i]= none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
 }
 }
 }
}
if (m_Figure.Direction[0]==right) //判断数字3
if (m_Figure.Direction[1]==down)
if (m_Figure.DotCount==1)
{
 if (m_Figure.Direction[2]==left)
 if (m_Figure.Direction[3]==right)
 {
 MessageBox("3");
 }
 for (int i =0; i<16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i]= none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
}
if (m_Figure.Direction[0]==down) //判断数字4
if (m_Figure.Direction[1]==right)
if (m_Figure.DotCount==2)
{
 if (m_Figure.Direction[2]==down)
 if (m_Figure.Direction[3]==none)
 {
 for (int i =0; i<16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i]= none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 MessageBox("4");
 return;
 }
}
if (m_Figure.Direction[0]==down) //判断数字5
if (m_Figure.Direction[1]==right)
if (m_Figure.DotCount==2)
if (m_Figure.Direction[2]==down)
if (m_Figure.Direction[3]==left)
if (m_Figure.Direction[4]==right)

```



```

 if (m_Figure.Direction[5]==none)
 {
 MessageBox("5");
 for (int i=0; i<16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i]= none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
 }

 if (m_Figure.DotCount==1) //判断数字6
 {
 if (m_Figure.Direction[1]==down)
 {
 if (m_Figure.Direction[2]==right)
 {
 if(m_Figure.Direction[3]==up)
 {
 if(m_Figure.Direction[4]==left)
 {
 MessageBox("6");
 for (int i=0; i<16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i]= none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
 }
 }
 }
 }
 }

 if (m_Figure.DotCount==1) //判断数字8
 {
 if (m_Figure.Direction[0]==left)
 {
 {
 if (m_Figure.Direction[1]==down)
 {
 if(m_Figure.Direction[2]==left)
 {
 if(m_Figure.Direction[3]==up)
 {
 if(m_Figure.Direction[4]==right)
 {
 if(m_Figure.Direction[5]==up)
 {
 MessageBox("8");
 for (int i=0; i<16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i]= none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
 }
 }
 }
 }
 }
 }
 }
 if (m_Figure.Direction[0]==down) //第2种判断数字8的方式
 {
 {
 if (m_Figure.Direction[1]==right)
 {
 if(m_Figure.Direction[2]==down)
 {
 if(m_Figure.Direction[3]==right)
 {
 if(m_Figure.Direction[4]==up)
 {
 if(m_Figure.Direction[5]==left)
 {
 MessageBox("8");
 for (int i=0; i<16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i]= none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
 }
 }
 }
 }
 }
 }
 }
 if (m_Figure.Direction[0]==left) //第3种判断数字8的方式
 {
 {
 if (m_Figure.Direction[1]==down)
 {
 if(m_Figure.Direction[2]==right)
 {
 if(m_Figure.Direction[3]==down)
 {
 if(m_Figure.Direction[4]==left)
 {
 if(m_Figure.Direction[5]==up)
 {
 MessageBox("8");
 for (int i=0; i<16; i++) //重新初始化m_Figure
 {

```

```

 m_Figure.Direction[i] = none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
}

}

}
if (m_Figure.DotCount==1) //判断数字9
{
 if (m_Figure.Direction[0]==left)
 {
 if (m_Figure.Direction[1]==down)
 if(m_Figure.Direction[2]==right)
 {
 MessageBox("9");
 for (int i=0; i<16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i] = none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
 }
 }
 if (m_Figure.Direction[0]==right) //第2种判断数字9的方式
 if (m_Figure.Direction[1]==up)
 if (m_Figure.Direction[2]==left)
 if (m_Figure.Direction[3]==down)
 {
 MessageBox("9");
 for (int i=0; i<16; i++) //重新初始化m_Figure
 {
 m_Figure.Direction[i] = none;
 }
 m_Figure.DotCount = 0;
 m_Panel.Invalidate(); //更新图片控件
 return;
 }
 }
}
for (int i =0; i<16; i++) //如果没有匹配的模板重新初始化m_Figure
{
 m_Figure.Direction[i]= none;
}
m_Figure.DotCount = 0;
m_Panel.Invalidate(); //更新图片控件
}

```

### 举一反三

根据本实例，读者可以：

- 实现手写汉字的识别。

## 3.7 图像字体

字体在图形、图像处理中具有举足轻重的作用，特殊的字体能够增加图像的显示效果。本节将通过几个实例介绍如何绘制特殊效果的字体。

### 实例 132 旋转的文字

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\03\132

#### 实例说明

在一些多媒体应用软件中，一些文字信息并不是按水平方向或垂直方向显示，而是按一定的角度倾斜显示，效果很好。本实例实现了文字的旋转，效果如图 3.39 所示。



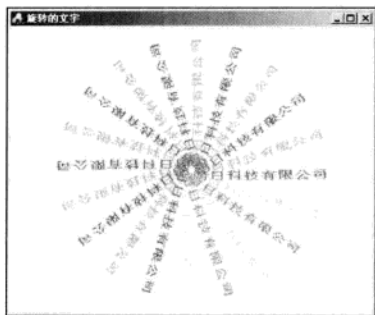


图 3.39 旋转的文字

## 技术要点

实现字体的旋转非常简单, 首先创建一个字体, 在创建字体时指定倾斜角度, 然后利用设备上下文选中字体, 最后输出文字, 这样文字就会在某一位置上按照字体指定的角度倾斜。创建一个字体可以调用 CFont 类的 CreateFont 方法, 语法如下:

```
BOOL CreateFont(int nHeight, int nWidth, int nEscapement, int nOrientation, int nWeight, BYTE bItalic, BYTE bUnderline, BYTE cStrikeOut, BYTE nCharSet, BYTE nOutPrecision, BYTE nClipPrecision, BYTE nQuality, BYTE nPitchAndFamily, LPCTSTR lpszFacename);
```

参数说明:

- nHeight、nWidth: 用于指定字体的高度和宽度。为正数时, 字体映射机制会根据指定的高度或宽度从字体列表中选择一种接近的字体, 以该字体的单元高度或宽度为参考。为负数时, 字体映射机制也会选择一种字体, 不过这时以字体字符高度或宽度为参考。单元是在实际输出字符的上下有一些空隙, 而字符是忽略掉这些空隙之后的单元高度。
- nEscapement: 以 x 轴为参考确定文本的倾斜角度。
- nOrientation: 确定字符基线与 x 轴的倾斜角度。
- nWeight: 确定字体的重量, 即字体的粗细程度。
- bItalic: 确定是否是斜体。
- bUnderline: 确定是否有下划线。
- cStrikeOut: 确定是否有删除线。
- nCharSet: 用于指定字符集。
- nOutPrecision: 确定字体映射机制如何根据提供的参数选择合适的字体。
- nClipPrecision: 用于确定字体的裁减精度。当文本的一部分延伸到指定区域外时, 该参数用于确定文本被裁减的方式。
- nQuality: 用于确定字体显示的品质。
- nPitchAndFamily: 用于确定字符间距和字体属性。
- lpszFacename: 确定字体名称。

例如实现一个简单的旋转文字, 代码如下:

```
CFont m_font;
m_font.CreateFont(-14,-10,i*10,0,600,0,0,0,DEFAULT_CHARSET,OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,FF_ROMAN,"宋体");
```

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 主要程序代码如下:

```
CDC* pDC = GetDC(); //获得设备上下文
CFont m_font;
pDC->SetBkMode(TRANSPARENT); //设置背景透明
CRect m_rect;
GetClientRect(m_rect); //获得客户区域
```

## 举一反三

### ● 修改任意控件的字体属性

实例位置: 光盘\mingrisoft\03\133

用过 Visual Basic、Delphi 的读者知道，在设置控件字体时，只要在控件的属性对话框中单击字体属性时，就会出现一个字体列表，供用户选择，使用起来比较方便。本实例实现了利用 Visual C++ 获取字体列表的功能，效果如图 3.40 所示。

图 3.40 当前系统字体列表

Windows 操作系统提供了一个 EnumFontFamiliesEx 函数，该函数用于获取系统字体列表，其语法如下：

参数说明:

- **hdc**: 是一个设备上下文句柄。
- **lpLogfont**: 是 LOGFONT 结构指针, 用于描述列举的字体信息。如果设置 LOGFONT 结构的 lfCharset 成员为 DEFAULT\_CHARSET, 函数将列举所有字符集的字体, 否则只列举指定的字符集字体。

- `lpEnumFontFamExProc`: 是列举字体的一个函数指针, 语法如下:

```
int CALLBACK EnumFontFamExProc(ENUMLOGFONTEX *lpelfe, NEWTEXTMETRICEX *lpntme,
int FontType,LPARAM lParam);
```

参数说明:

- **lpelfe**: 是一个 **ENUMLOGFONTEX** 结构指针, 该结构包含了列举的字体信息。
- **lpntme**: 是一个 **NEWTEXTMETRICEX** 结构指针, 该结构包含了列举的字体物理信息。
- **FontType**: 描述了列举到的字体类型。
- **lParam**: 是由 **EnumFontFamiliesEx** 函数传递的附加信息。

实际上 EnumFontFamiliesEx 函数不断地调用 lpEnumFontFamExProc 函数指针所指的函数，向其参数传递字体信息。用户只要定义一个与 lpEnumFontFamExProc 具有相同函数原型的函数，在该函数中读取 lpelfe 参数的 lfFaceName 成员，将其存储在一个列表中，就可以获取字体名称了。lParam 是应用程序定义的一个附加信息。dwFlags 是保留字，必须为零。

通过了解 EnumFontFamiliesEx 函数,读者会得到很多启发,函数指针可以作为函数参数,这样的函数具有很大的弹性。因为函数指针所指的函数由用户自己定义,同一个函数由于参数



函数指针所指的函数不同,实现的功能也不同。类似的函数在 Windows 系统中有很多,例如 CreateThread 函数。在创建一个线程时,线程需要执行一个函数(线程函数),在设计 CreateThread 函数时,参数中就提供了一个函数指针,由用户定义线程函数的行为,将其传递给函数指针参数,在 CreateThread 函数内部就可以访问线程函数了。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框类中放置静态文本和组合框控件。
- (3) 定义一个回调函数,用于获取列举的字体名称,代码如下:

```
CStringList fontlist;
int CALLBACK EnumFontList (const ENUMLOGFONTEX *lpelfe, const NEWTEXTMETRICEX *lpntme,
unsigned long FontType, LPARAM lParam)
{
 POSITION pos = fontlist.Find(lpelfe->elfLogFont.lfFaceName); //获得字体名称
 if (pos == NULL)
 fontlist.AddTail(lpelfe->elfLogFont.lfFaceName);
 return 1;
}
```

- (4) 调用 EnumFontFamiliesEx 函数列举字体,代码如下:

```
CDC* dc = GetDC();
CString str;
fontlist.RemoveAll();
m_FontList.ResetContent();
LOGFONT m_logfont;
memset(&m_logfont,0,sizeof(m_logfont));
m_logfont.lfCharSet = DEFAULT_CHARSET;
m_logfont.lfFaceName[0] = NULL;
EnumFontFamiliesEx(dc->m_hDC,&m_logfont,(FONTENUMPROC)EnumFontList,100,0); //枚举系统字体
POSITION pos;
for (pos = fontlist.GetHeadPosition(); pos != NULL;)
{
 str = fontlist.GetNext(pos);
 m_FontList.AddString(str); //将字体名称插入到组合框中
}
```

## 举一反三

根据本实例,读者可以:

- 列举系统中的各种资源。

## 实例 134 空心文字

本实例是一个人性化的实例

实例位置: 光盘\mingrisoft\03\134

## 实例说明

Windows 提供了一个空心的字体——华文彩云,该字体在设备上输出时是空心的,效果比较好。本实例实现了宋体空心文本,效果如图 3.41 所示。

## 技术要点

实现字体的空心显示,可以利用设备上下文 CDC 类的通道方法。CDC 类提供了多个通道方法,其主要方法如下:

- (1) BeginPath。该方法用于开始一个通道,语法如下:  
BOOL BeginPath();
- (2) EndPath。该方法用于结束一个通道,语法如下:  
BOOL EndPath();
- (3) StrokePath。该方法用当前的画笔绘制一个通道,语法如下:

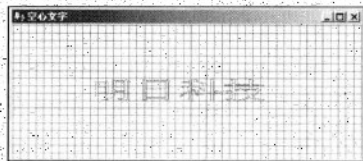


图 3.41 空心文字

BOOL StrokePath();

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框类中添加 DrawForm 方法绘制网格，代码如下：

```
void CPathFontDlg::DrawForm()
{
 CDC* dc = GetDC(); //获得设备上下文
 CRect m_rect;
 GetClientRect(m_rect); //获得客户区域
 CBrush brush;
 brush.CreateStockObject(WHITE_BRUSH); //创建画刷
 dc->FillRect(m_rect,&brush); //填充区域
 CPen m_pen(PS_SOLID,1,RGB(63,126,126)); //创建画笔
 dc->SelectObject(&m_pen);
 for (int x = 1 ; x<m_rect.right;x+=10) //绘制竖线
 {
 dc->MoveTo(x,0);
 dc->LineTo(x,m_rect.bottom);
 }
 for (int y = 1 ; y<m_rect.bottom;y+=10) //绘制横线
 {
 dc->MoveTo(0,y);
 dc->LineTo(m_rect.right,y);
 }
}
```

(3) 绘制空心文字，代码如下：

```
CDC* pDC = GetDC();
CRect m_rect;
GetClientRect(m_rect);
m_font.CreateFont(-32,-28,0,0,600,0,0,0,DEFAULT_CHARSET,OUT_DEFAULT_PRECIS,
 CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,FF_ROMAN,"宋体"); //创建字体
pDC->BeginPath(); //打开通道
pDC->SelectObject(&m_font);
pDC->SetBkMode(TRANSPARENT); //设置背景透明
pDC->TextOut(100,70,"明日科技"); //输出字体
CPen pen(PS_SOLID,1,RGB(255,0,255)); //设置画笔
pDC->SelectObject(&pen);
pDC->EndPath(); //关闭通道
m_font.Detach(); //释放字体
pDC->StrokePath();
```

## 举一反三

根据本实例，读者可以：

- 设计字型窗体。

## 实例 135 彩虹文字

这是一个可以提高分析能力的实例

实例位置：光盘\mingrisoft\03\135

## 实例说明

在窗体中显示彩虹文字，可以美化程序界面。本实例实现的是在窗体中显示彩虹文字，运行程序如图 3.42 所示。

## 技术要点

在窗体中显示彩虹文字，需要调用 CDC 类的 BeginPath、EndPath 和 AbortPath 等方法使用通道，并使用随机函数 rand 来设置颜色。

## 实现过程

(1) 新建一个基于对话框的应用程序。

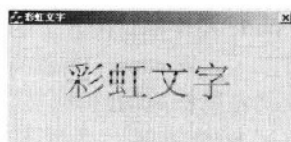


图 3.42 彩虹文字



(2) 主要程序代码如下:

```
void CRainbowDlg::OnTimer(UINT nIDEvent)
{
 CDC* pDC = GetDC(); //获得设备上下文
 Font.CreatePointFont(400,"宋体",pDC); //创建字体
 pDC->SelectObject(&Font);
 pDC->BeginPath(); //打开通道
 pDC->SetBkMode(TRANSPARENT); //设计背景透明
 pDC->TextOut(70,45,"彩虹文字"); //输出文字
 pDC->EndPath(); //关闭通道
 pDC->SelectClipPath(RGN_COPY);
 pDC->AbortPath();
 Font.DeleteObject();
 CRect rect;
 GetClientRect(&rect);
 int R,G,B;
 for(int i=0;i<rect.Height();i=i+5)
 {
 //随机选择颜色
 R = rand()/2;
 G = rand()/2;
 B = rand()/2;
 CPen pen;
 pen.CreatePen(PS_SOLID,5,RGB(255*R,255*G,255*B)); //创建画笔
 pDC->SelectObject(&pen);
 pDC->MoveTo(rect.Width(),i);
 pDC->LineTo(0,i);
 pen.DeleteObject();
 }
 CDialog::OnTimer(nIDEvent);
}
```

### 举一反三

根据本实例,读者可以:

- 实现跟随鼠标移动的图片。

## 实例 136

### 如何在图片上平滑移动文字

这是一个可以提高分析能力的实例

实例位置: 光盘\mingrisoft\03\136

### 实例说明

在图片界面中,可以根据需要在图片上移动文字或一些卡通图片,使图片更具有个性化和动态性。本实例实现的是在图片上移动文字,如图 3.43 所示,并且文字可以移出图片的显示范围。

### 技术要点

在图片上移动文字比较简单,可以利用 CStatic 控件标识图片上的文字信息。在鼠标拖动文字时,只需要调整 CStatic 控件的位置就可以了。Windows 并没有提供鼠标拖动的消息,但是可以根据鼠标拖动的开始时间和生存期确定鼠标拖动消息。鼠标拖动消息的起点应该在鼠标按下并开始移动时,终点在用户释放鼠标按钮时。为了标识鼠标是否处于拖动状态,可以定义一个布尔型成员变量 m\_IsDowned,在用户按下鼠标按钮时,将其设置为 TRUE,表示开始鼠标拖动,在用户释放鼠标按钮时,将其设置为 FALSE,表示结束鼠标拖动。在鼠标移动过程中,判断 m\_IsDowned 是否为 TRUE,如果是则表明用户正在进行拖动操作,此时可以移动 CStatic 控件。这样就实现了 CStatic 控件的拖动。



图 3.43 如何在图片上平滑移动文字

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加静态文本控件和图片控件。
- (3) 从 CStatic 类派生一个子类 CMyStatic, 在该类中定义一个变量 m\_font, 用于设置文本的字体。
- (4) 处理 CMyStatic 的 WM\_PAINT 消息, 绘制文本, 代码如下:

```
void CMyStatic::OnPaint()
{
 CPaintDC dc(this);
 CDC* pDC = GetDC();
 pDC->SetBkMode(TRANSPARENT); //设置背景透明
 pDC->SelectObject(&m_font); //选入字体
 pDC->SetTextColor(RGB(255,0,0)); //设置文本颜色
 CString str;
 this->GetWindowText(str); //获得显示文本
 pDC->TextOut(0,2,str); //绘制文本
}
```

- (5) 处理对话框的 WM\_LBUTTONDOWN 消息, 记录鼠标按下时的坐标, 代码如下:

```
void CMoveTextDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
 CRect m_client;
 m_title.GetClientRect(m_client); //获得客户区域
 m_title.MapWindowPoints(this,m_client); //转为屏幕区域
 if (m_client.PtInRect(point)) //如果鼠标在文本区域内
 m_IsDowned = TRUE;
 m_start = point; //设置当前鼠标位置为起点
 //设置终点坐标
 CRect m_rect1;
 m_title.GetClientRect(m_rect1);
 m_title.MapWindowPoints(this,m_rect1);
 m_end.x = m_rect1.left;
 m_end.y = m_rect1.top;
 CDialog::OnLButtonDown(nFlags, point);
}
```

- (6) 处理对话框的 WM\_MOUSEMOVE 消息, 判断是否拖动控件, 如果拖动控件, 根据当前鼠标点移动控件, 代码如下:

```
void CMoveTextDlg::OnMouseMove(UINT nFlags, CPoint point)
{
 if (m_IsDowned)
 {
 m_IsMove = TRUE;
 int x,y;
 x = point.x-m_start.x;
 y = point.y-m_start.y;
 m_title.GetClientRect(m_rect);
 x+= m_end.x;
 y+= m_end.y;
 m_rect.DeflateRect(x, y,0,0);
 m_rect.InflateRect(0,0,x,y);
 this->InvalidateRect(m_rect); //重绘文本区域
 m_title.MoveWindow(m_rect); //移动控件
 }
 CDialog::OnMouseMove(nFlags, point);
}
```

- (7) 处理对话框的 WM\_LBUTTONUP 消息, 结束鼠标拖动。代码如下:

```
void CMoveTextDlg::OnLButtonUp(UINT nFlags, CPoint point)
{
 m_IsDowned = FALSE;
 m_IsMove = FALSE;
 CDialog::OnLButtonUp(nFlags, point);
}
```

## 举一反三

根据本实例, 读者可以:

- 实现跟随鼠标移动的图片。



## 实例 137 图像水印效果

这是一个可以提高分析能力的实例

实例位置: 光盘\mingrisoft\03\137

## 实例说明

给图片添加水印是指在图片上附加指定的文字。现在许多软件或网页中的图片都有水印, 水印并不是图片本身的信息, 而是在图片加载时动态附加上去的。本实例实现的是在图片上添加水印, 运行程序如图 3.44 所示。

## 技术要点

本实例使用 GDI+ 所提供的功能来实现水印的添加, GDI+ 是 GDI 图形库的一个增强版本, Visual C++ 可以使用这个库。它内建于 Windows XP 和 Microsoft .NET, 而对于 Windows 98、Windows NT 和 Windows 2000, 则有一个可重新发布的版本。GDI+ 是一个 Visual C++ API。它使用 Visual C++ 类和 Visual C++ 方法。为了使用 GDI+, 必须包含 `(#include) <gdiplus.h>` 文件, 并将工程链接到 `gdiplus.lib` 库, 这两个文件包含在最新的 Windows SDK 中。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加 4 个编辑框控件, 分别用来显示附加水印图片的路径、水印文本、水印在图片中的  $x$  坐标与  $y$  坐标。添加两个按钮控件, 分别设置其 `Caption` 属性为“字体”和“水印处理”。添加两个复选框控件, 分别设置其 `Caption` 属性为“水平居中”和“垂直居中”。

- (3) 定义名为 `GetCodecClsid` 的全局函数, 该函数用来获取指定文件类型的唯一标识。代码如下:

```
int GetCodecClsid(const WCHAR* format, CLSID* pClsid)
{
 UINT codenum = 0;
 UINT size = 0;
 ImageCodecInfo* pImageCodecInfo = NULL;
 GetImageEncodersSize(&codenum, &size);
 if(size == 0)
 return -1;
 pImageCodecInfo = new ImageCodecInfo[size];
 if(pImageCodecInfo == NULL)
 return -1;
 GetImageEncoders(codenum, size, pImageCodecInfo);
 for(UINT j = 0; j < codenum; ++j)
 {
 if(wcsncmp(pImageCodecInfo[j].MimeType, format) == 0)
 {
 *pClsid = pImageCodecInfo[j].Clsid;
 delete []pImageCodecInfo;
 return 0;
 }
 }
 delete []pImageCodecInfo;
 return -1;
}
```

- (4) 单击窗体上的“水印处理”按钮实现对图片的水印添加。代码如下:

```
void CWaterMarkerDlg::OnMarkerhandle()
{
 //获得控件中数据
 CString szFileDir, szMarkerText, szHorX, szVerY;
 m_FileDir.GetWindowText(szFileDir);
 m_MarkerText.GetWindowText(szMarkerText);
 m_VerY.GetWindowText(szVerY);
 m_HorX.GetWindowText(szHorX);
```

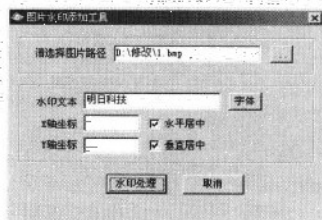


图 3.44 图像水印效果

```

if (!szFileDir.IsEmpty())
{
 if (!m_bSetFont) //如果没有设置字体, 先设置字体
 {
 OnBtFont();
 }
 try
 {
 //格式化坐标
 int nX = atoi(szHorX.GetBuffer(0));
 szHorX.ReleaseBuffer();
 int nY = atoi(szVerY.GetBuffer(0));
 szVerY.ReleaseBuffer();
 //指定文件类型标识
 CLSID clsid;
 GetCodecClsid(L"image/jpeg", &clsid);
 Brush *brush = new SolidBrush (Color(m_Red, m_Green, m_Blue));
 Font *font = new Font(GetDC()->m_hDC, &m_LogFont);
 PointF ptf;
 int nLen = MultiByteToWideChar(CP_ACP, 0, szMarkerText, -1, NULL, 0);
 int nQuality = 95;
 EncoderParameters Encoders;
 Encoders.Count = 1;
 Encoders.Parameter[0].Guid = EncoderQuality;
 Encoders.Parameter[0].Type = EncoderParameterValueTypeLong;
 Encoders.Parameter[0].NumberOfValues = 1;
 Encoders.Parameter[0].Value = &nQuality;
 //判断是否为位图
 if (m_Strextend == "jpg" || m_Strextend == "jpeg" || m_Strextend == "bmp")
 {
 Bitmap *pBmp = Bitmap::FromFile(szFileDir.AllocSysString());
 if (pBmp)
 {
 Graphics *pGraph = Graphics::FromImage(pBmp);
 //获取字符串的宽度
 PointF origin(0.0f, 0.0f);
 RectF TextRC;
 pGraph->MeasureString(szMarkerText.AllocSysString(), nLen, font, origin, &TextRC);
 szMarkerText.ReleaseBuffer();
 //设置文本位置
 CButton* pHorButton = (CButton*)GetDlgItem(IDC_HORCENTER);
 if (!pHorButton->GetCheck())
 {
 ptf.X = nX;
 }
 else //水平方向居中
 {
 //获取图像宽度
 int nWidth = pBmp->GetWidth();
 int nChar = TextRC.Width;
 //设置文本的水平方向位置
 ptf.X = (nWidth - nChar) / 2;
 }
 CButton* pVerButton = (CButton*)GetDlgItem(IDC_VERCENTER);
 if (!pVerButton->GetCheck())
 {
 ptf.Y = nY;
 }
 else //垂直方向居中
 {
 //获取图像高度
 int nHeight = pBmp->GetHeight();
 int nCharHeight = TextRC.Height;
 //设置文本的水平方向位置
 ptf.Y = (nHeight - nCharHeight) / 2;
 }
 pGraph->DrawImage(pBmp, 0, 0, pBmp->GetWidth(), pBmp->GetHeight());
 pGraph->DrawString(szMarkerText.AllocSysString(), nLen, font, ptf, brush);
 //创建文件夹并设置添加水印后的图片路径
 szMarkerText.ReleaseBuffer();
 int pos = szFileDir.ReverseFind('\\');
 char chName[MAX_PATH] = {0};
 strcpy(chName, szFileDir.Left(pos));
 strcat(chName, "\\JPG");
 CreateDirectory(chName, NULL);
 CString JpgFile = chName;
 }
 }
 }
}

```

```

 JpgFile += "\\";
 JpgFile += m_FileName;
 //保存图片
 pBmp->Save(JpgFile.AllocSysString(), &clsid, &Encoders);
 JpgFile.ReleaseBuffer();
 delete pGraph;
 }
 delete pBmp;
 MessageBox("添加成功!");
}
}
catch(...)
{
 MessageBox("添加失败!");
}
}
}

```

## 举一反三

根据本实例，读者可以：

● 为图片批量添加水印。

### 3.8 图像管理

### 实例 138

## 管理计算机内图片文件的程序

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\03\138

## 实例说明

如今的应用软件在满足用户需求的同时,通常还增加许多额外的功能。例如,添加媒体播放工具、磁盘管理工具等。本实例提供了一个图片管理工具,能够将用户某一路径下的位图文件提取出来,并保存到指定的路径下,效果如图 3.45 所示。

## 技术要点

本实例中需要解决如下问题。

- (1) 列举磁盘目录。
- (2) 遍历某一个磁盘下的所有文件夹。
- (3) 获取某一文件夹下的所有位图文件。
- (4) 复制位图文件。

对于问题 (1) 列举磁盘目录, 可以通过 `GetLogicalDriveStrings` API 函数实现, 该函数语法如下:

```
DWORD GetLogicalDriveStrings(DWORD nBufferLength, LPTSTR lpBuffer);
```

参数说明:

- `nBufferLength`: 标识缓冲区的大小。
- `lpBuffer`: 标识一个缓冲区。函数会将磁盘信息返回到 `lpBuffer` 中。
- 返回值: 实际复制到缓冲区中的字符长度。

在实际应用中，通常用户并不知道盘符共占用多少空间，因此可以按如下方式调用 `GetLogicalDriveStrings` 函数获取缓冲区占用的空间。

DWORD dirlen = GetLogicalDriveStrings(0, NULL);

根据函数的返回值确定缓冲区大小，再次调用 `GetLogicalDriveStrings` 函数获取磁盘信息，代码如下：



图 3.45 管理计算机内图片文件的程序

```
LPSTR pdir, phead;
pdir = new char [dirilen+1];
GetLogicalDriveStrings(dirilen, pdir);
phead = pdir;
while (*pdir != 0)
{
 m_disk.AddString(pdir);
 pdir = _tcschr(pdir, 0) + 1;
}
delete[] phead;
```

对于问题(2) 遍历某一个磁盘下的所有文件夹, 实现起来较为复杂, 需要使用 FindFirstFile 函数和 FindNextFile 函数。

(1) FindFirstFile。该函数用于查找某一个目录下的第一个文件。语法如下:

```
HANDLE FindFirstFile(LPCTSTR lpFileName, LPWIN32_FIND_DATA lpFindFileData);
```

参数说明:

- lpFileName: 标识查找的文件名, 可以包含通配符。
- lpFindFileData: 是 WIN32\_FIND\_DATA 结构指针, 用于存储找到的文件信息。
- 返回值: 函数返回一个查找句柄, 用于 FindFirstFile 函数。

(2) FindNextFile。该函数根据查找句柄查找下一个文件。语法如下:

```
BOOL FindNextFile(HANDLE hFindFile, LPWIN32_FIND_DATA lpFindFileData);
```

参数说明:

- hFindFile: 是查找句柄, 通常为 FindFirstFile 函数的返回值。
- lpFindFileData: 是 WIN32\_FIND\_DATA 结构指针, 用于存储找到的文件信息。

使用 FindFirstFile 函数和 FindNextFile 函数只能实现一次查找, 要遍历整个磁盘, 需要编写递归函数实现, 详细代码可参考实现过程。

问题(2) 解决了, 问题(3) 获取某一文件夹下的所有位图文件也就迎刃而解了, 只是在查找文件时, 需要明确指定文件扩展名。详细代码可参考实现过程。

对于问题(4) 复制位图文件, 利用 CopyFile API 函数就可实现了。语法如下:

```
BOOL CopyFile(LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName, BOOL bFailIfExists);
```

参数说明:

- lpExistingFileName: 标识预复制的文件。
- lpNewFileName: 标识新的文件名称。
- bFailIfExists: 确定如果目标文件存在是否进行替换。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框中添加树控件、组合框控件、列表视图控件和按钮控件。

(3) 向对话框类中添加 EnumDIR 方法, 用于遍历指定磁盘下的目录, 代码如下:

```
void CManagemgeDlg::EnumDIR(CString dirname, HTREEITEM hparentitem)
{
 WIN32_FIND_DATA m_fileinfo; //记录查找到的文件信息
 HANDLE hfile; //查找句柄
 HTREEITEM hnode; //树节点句柄
 CString temp = dirname;
 CString tempfile;
 dirname += "*.*"; //查找所有文件
 hfile = FindFirstFile(dirname, &m_fileinfo);
 //如果是目录, 继续查找
 if (m_fileinfo.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
 {
 tempfile = m_fileinfo.cFileName;
 tempfile.TrimLeft();
 tempfile.TrimRight();
 if ((tempfile != ".") && (tempfile != ".."))
 {
 hnode = m_tree.InsertItem(m_fileinfo.cFileName, 0, 0, hparentitem); //将查找的目录插入到树控件中
 }
 if (m_fileinfo.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
 {
 EnumDIR(tempfile + dirname, hnode);
 }
 }
}
```



```

 if ((tempfile != ".") && (tempfile != ".."))
 {
 EnumDIR(temp+"\\ "+m_fileinfo.cFileName+"\\",hnode);
 }
 }
 while (FindNextFile(hfile,&m_fileinfo))
 {
 if (m_fileinfo.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
 {
 tempfile = m_fileinfo.cFileName;
 tempfile.TrimLeft();
 tempfile.TrimRight();
 if ((tempfile != ".") && (tempfile != ".."))
 {
 hnode = m_tree.InsertItem(tempfile,0,0,hparentitem);
 }
 if (m_fileinfo.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
 if ((tempfile != ".") && (tempfile != ".."))
 {
 EnumDIR(temp+"\\ "+m_fileinfo.cFileName,hnode);
 }
 }
 }
}

```

(4) 向对话框类中添加 EnumFiles 方法, 列举指定目录下的所有位图文件, 代码如下:

```

void CManageImageDlg::EnumFiles(LPCTSTR filepath)
{
 WIN32_FIND_DATA m_fileinfo; //记录查找到的文件信息
 HANDLE hfile; //文件查找句柄
 m_dir.ResetContent();
 CString tempfile;
 tempfile = filepath;
 tempfile = tempfile.Left(tempfile.GetLength()-3);
 CString c_temp,c_fileext;
 m_disk.GetWindowText(c_temp);
 //查找文件
 hfile = FindFirstFile(filepath,&m_fileinfo);
 if (!(m_fileinfo.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
 {
 c_temp = m_fileinfo.cFileName;
 c_fileext = c_temp.Right(3);
 if (c_fileext=="bmp")
 m_dir.AddString(tempfile+m_fileinfo.cFileName);
 }
 while (FindNextFile(hfile,&m_fileinfo))
 {
 //如果找到的文件是一个目录, 继续查找子目录
 if (!(m_fileinfo.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
 {
 c_temp = m_fileinfo.cFileName;
 c_fileext = c_temp.Right(3);
 if (c_fileext=="bmp")
 m_dir.AddString(tempfile+m_fileinfo.cFileName);
 }
 }
}

```

(5) 处理“保存”按钮的单击事件, 将所有位图文件保存到某一目录下, 代码如下:

```

void CManageImageDlg::OnSave()
{
 int m_count = m_dir.GetCount();
 if (m_count>0)
 {
 //构造另存为对话框
 CFileDialog m_fdlg(FALSE,"bmp",NULL, OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,NULL,this);
 if (m_fdlg.DoModal()==IDOK)
 {
 CString m_savepath = m_fdlg.GetPathName(); //获得文件路径
 CString m_curfile;
 for (int i = 0;i < m_count;i++)
 {
 m_dir.GetText(i,m_curfile);
 //复制文件
 CopyFile(m_curfile,ExtractFileName(m_savepath)+"\\"+ ExtractFilePath(m_curfile),FALSE);
 }
 MessageBox("保存成功");
 }
 }
}

```

## 举一反三

根据本实例，读者可以：

- 实现文件查找。

## 实例 139

## 提取并保存应用程序图标

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\03\139

## 实例说明

许多应用软件都具有漂亮的应用程序图标。本实例实现了提取并保存应用程序图标的功能，效果如图 3.46 所示。

## 技术要点

提取应用程序的图标比较简单，可以利用 ExtractIcon API 函数实现，该函数语法如下：

```
HICON ExtractIcon(HINSTANCE hInst, LPCTSTR lpszExeFileName, UINT nIndex);
```

参数说明：

- hInst：是当前应用程序的实例句柄。
- lpszExeFileName：标识可执行文件的名称。
- nIndex：标识返回的图标索引，如果为零，将返回所标识文件的第一个图标句柄。
- 返回值：提取的图标句柄。

保存应用程序图标略微复杂，需要了解图标的结构。图标文件由 3 部分组成，第 1 部分是图标文件头，结构如下：

```
typedef struct
{
 WORD idReserved; // 保留 (必须是 0)
 WORD idType; // 资源类型 (1标识ICON)
 WORD idCount; // 文件所含图片的数量
 ICONDIRENTRY idEntries[1]; // 每一个图片的入口
} ICONDIR, *LPICONDIR;
```

一个图标文件可以包含多个图标，成员 idCount 用于确定文件包含的图标数量。

第 2 部分是图标资源索引目录，结构如下：

```
typedef struct
{
 BYTE bWidth; // 图像宽度 (单位是像素)
 BYTE bHeight; // 图像高度 (单位是像素)
 BYTE bColorCount; // 图像中的颜色数目 (0 if >= 8bpp)
 BYTE bReserved; // 保留 (必须是0)
 WORD wPlanes; // 颜色平面
 WORD wBitCount; // 位深
 DWORD dwBytesInRes; // 这幅图片所占用的字节数目
 DWORD dwImageOffset; // 图片在文件中的位置
} ICONDIRENTRY, *LPICONDIRENTRY;
```

其中 dwBytesInRes 成员用于确定图标的大小，dwImageOffset 确定图标数据在文件中的位置。如果图标文件中包含多个图标，则每一个图标都对应一个图标资源索引目录。

第 3 部分是图标数据，结构如下：

```
typedef struct
{
 BITMAPINFOHEADER icHeader; // DIB 信息头
 RGBQUAD icColors[1]; // 颜色表
 BYTE icXOR[1]; // XOR掩码
 BYTE icAND[1]; // AND掩码
} ICONIMAGE, *LPICONIMAGE;
```

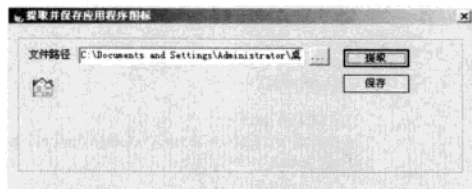


图 3.46 提取并保存应用程序图标

其中 icHeader 成员是位图信息头, icColors[1]成员标识颜色表, 大小由 icHeader 成员确定。  
icXOR[1]成员标识图标 XOR 掩码的 DIB 位, icAND[1]成员标识图标 AND 单色掩码。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中放置编辑框控件、按钮控件、静态文本控件和图片控件。
- (3) 在对话框类的头文件中定义图标文件结构。
- (4) 处理“提取”按钮的单击事件, 提取应用程序图标, 代码如下:

```
//提取应用程序图标
void CFetchAndSaveIconDlg::OnFetch()
{
 CString str;
 m_filename.GetWindowText(str);
 if (!str.IsEmpty())
 {
 HICON m_hicon;
 m_hicon = ::ExtractIcon(AfxGetInstanceHandle(), str, 0); //获得图标句柄
 if (m_hicon != NULL)
 {
 m_demoicon.SetIcon(m_hicon); //显示图标
 }
 }
}
```

- (5) 处理“保存”按钮单击事件, 将提取的图标资源保存为图标文件, 代码如下:

```
void CFetchAndSaveIconDlg::OnSave()
{
 CFileDialog m_savedlg (FALSE, "ico", NULL, NULL, "图标(.ico)*.ico", this); //构造另存为对话框
 if (m_savedlg.DoModal() == IDOK)
 {
 CString str = m_savedlg.GetPathName(); //获得文件路径
 if (!str.IsEmpty())
 {
 CFile m_file (str, CFile::modeCreate | CFile::typeBinary | CFile::modeWrite); //构造文件对象
 HICON hicon;
 CString name;
 m_filename.GetWindowText(name); //获得文件名
 HMODULE hmodule = LoadLibraryEx(name, NULL, LOAD_LIBRARY_AS_DATAFILE);
 EnumResourceNames(hmodule, RT_GROUP_ICON, (ENUMRESNAMEPROC)EnumResNameProc, LONG(GetSafeHwnd()));
 hicon = (HICON)FindResource(hmodule, m_iconname, RT_GROUP_ICON); //查找图标资源
 HGLOBAL global = LoadResource(hmodule, (HRSRC)hicon); //加载图标资源
 if (global != NULL)
 {
 m_lpMemDir = (LPMEMICONDIR)LockResource(global); //锁定资源
 }
 lpicondir temp = (lpicondir)m_lpMemDir;
 m_lpdir = (lpicondir)m_lpMemDir;
 DWORD factsize;
 //写入文件头
 WORD a = m_lpdir->idreserved;
 m_file.Write(&a, sizeof(WORD));
 a = m_lpdir->idtype;
 m_file.Write(&a, sizeof(WORD));
 a = m_lpdir->idcount;
 m_file.Write(&a, sizeof(WORD));
 m_lpdir = NULL;
 //写入索引目录
 icondirent entry;
 for (int i = 0; i < temp->idcount; i++)
 {
 DWORD size;
 DWORD imagesize = GetImageOffset(hmodule, i, size);
 free(m_lpData);
 entry.bheight = m_lpMemDir->idEntries[i].bHeight;
 entry.bwidth = m_lpMemDir->idEntries[i].bWidth;
 entry.breserved = 0;
 entry.bcolorcount = m_lpMemDir->idEntries[i].bColorCount;
 entry.dwbytesinres = m_lpMemDir->idEntries[i].dwBytesInRes;
 entry.dwimageoffset = imagesize;
 entry.wbitcount = m_lpMemDir->idEntries[i].wBitCount;
 }
 }
 }
}
```

```

 entry.wplanes = m_lpMemDir->idEntries[i].wPlanes;
 m_file.Write(&entry,sizeof(entry));
 }
 //写入图像数据
 for (int j = 0; j < temp->idcount; j++)
 {
 LPBYTE pInfo;
 DWORD size;
 DWORD imagesize= GetImageOffset(hmodule,j,size,pInfo);
 m_file.Write(LPBYTE)m_lpData,size);
 free(m_lpData);
 }
 UnlockResource(global);
 FreeLibrary(hmodule);
 m_file.Close();
}
}
}

```

### 举一反三

根据本实例，读者可以：

- 提取图标文件中包含的所有图标。

## 3.9 图片动画

图片动画在编写多媒体应用程序中应用非常广泛，本节将通过制作屏保程序和图片动画等实例来讲解图片动画的制作方法。

### 实例 140

### 利用图片制作屏幕保护程序

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrsoft\03\140

### 实例说明

屏幕保护是当用户长时间没有对计算机作任何操作时，系统为了保护屏幕而启动的一种保护措施。屏幕保护启动后都会占据整个屏幕，并且具有一些缓慢变动的图像效果。运行程序，保持鼠标指针不动，计算机屏幕将进入屏幕保护界面，移动鼠标，即可取消屏幕保护，效果如图 3.47 所示。

### 技术要点

在程序运行的时候，利用 `GetSystemMetrics` 获得屏幕大小，使用 `MoveWindow` 函数全屏显示程序，通过函数 `GetCursorPos` 获取当前鼠标的坐标值，同时在移动鼠标时触发窗体的 `WM_MOUSEMOVE` 事件，在该事件下再次通过 `GetCursorPos` 函数获取当前鼠标的坐标值，与窗体启动时获取的鼠标坐标值进行比较，如果相同则不退出程序，不同则退出程序。

在定时器中实现图片的位置移动，从而实现屏保的效果。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个图片控件。
- (3) 向工程中导入一个位图资源。



图 3.47 利用图片制作屏幕保护程序



(4) 程序全屏显示, 获得鼠标当前位置并设置定时器, 代码如下:

```
int x,y;
//获得屏幕分辨率
x = GetSystemMetrics(SM_CXSCREEN);
y = GetSystemMetrics(SM_CYSCREEN);
this->MoveWindow(0,0,x,y); //全屏显示
GetCursorPos(&pOint); //获得鼠标位置
i = 0;
j = 0;
SetTimer(1,100,NULL);
```

(5) 设置图片移动, 代码如下:

```
void CScreenProtectDlg::OnTimer(UINT nIDEvent)
{
 CRect rect,rc;
 GetWindowRect(rc);
 m_picture.GetClientRect(rect); //获得图片控件的客户区域
 if(i<rc.Width())
 if(j<rc.Height())
 {
 m_picture.MoveWindow(i,j,rect.Width(),rect.Height(),true); //移动图片控件
 i += 5;
 j += 5;
 }
 else
 j=0;
 else
 i=0;
 CDialog::OnTimer(nIDEvent);
}
```

### 举一反三

根据本实例, 读者可以:

- 制作复杂的屏保程序;
- 制作电子相册程序。

## 实例 141 图片动画

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\03\141

### 实例说明

在一些应用程序的界面中, 如果窗体的背景图案在不停的变换, 一定会吸引用户的注意。运行程序, 在窗体上就会自动显示动画效果, 如图 3.48 所示。

### 技术要点

要实现背景图案的动画效果, 就要让背景上的图案不停地变换, 所以在实例中使用定时器不停地改变窗体背景。本实例中背景是使用代码绘制的一种图案, 应将改变窗体背景的代码添加到 WM\_TIMER 事件中, 从而达到动画背景的效果。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个图片控件。
- (3) 计算绘制图形用的各个点, 代码如下:

```
lpi=0;
arrays[0] = CPoint(148,0);
n=40;
//设置多边形顶点坐标
for(int i=1;i<n;i++)
{
```

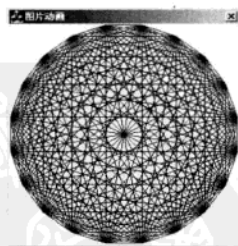


图 3.48 图片动画

```

lpi+=(2*PI/n);
if(lpi<=2*PI/4)
{
 arrays[i] = CPoint(148+148*sin(2*i*PI/n),148-148*cos(2*i*PI/n));
}
if(lpi>2*PI/4 && lpi<=2*PI/2)
{
 arrays[i] = CPoint(148+148*sin(PI-2*i*PI/n),148+148*cos(PI-2*i*PI/n));
}
if(lpi>2*PI/2 && lpi<=2*PI*3/4)
{
 arrays[i] = CPoint(148-148*sin(2*i*PI/n-2*PI/2),148+148*cos(2*i*PI/n-2*PI/2));
}
if(lpi>2*PI*3/4 && lpi<=2*PI)
{
 arrays[i] = CPoint(148-148*sin(2*PI-2*i*PI/n),148-148*cos(2*PI-2*i*PI/n));
}
}
result = true;
SetTimer(1,300,NULL); //开启定时器

```

(4) 绘制图形,代码如下:

```

void CPictureCartoonDlg::OnTimer(UINT nIDEvent)
{
 CRect rc;
 m_picture.GetClientRect(&rc); //获得控件客户区域
 CDC* pDC = m_picture.GetDC(); //获得设备上下文
 pDC->FillRect(&rc,NULL); //填充设备上下文
 CPen pen;
 pen.CreatePen(PS_SOLID,1,RGB(0,0,255)); //创建画笔
 pDC->SelectObject(&pen);
 //交替绘制两种多边形
 if(result)
 {
 for(int j=0;j<n;j=j+2)
 {
 for(int k=0;k<=j;k=k+2)
 {
 pDC->MoveTo(arrays[j]);
 pDC->LineTo(arrays[k]);
 }
 }
 }
 else
 {
 for(int j=1;j<=n;j=j+2)
 {
 for(int k=1;k<=j;k=k+2)
 {
 pDC->MoveTo(arrays[j]);
 pDC->LineTo(arrays[k]);
 }
 }
 }
 result = !result;
 CDialog::OnTimer(nIDEvent);
}

```

### 举一反三

根据本实例,读者可以:

- 在窗体的背景上绘制图片。

## 实例 142 指法练习软件

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\03\142

### 实例说明

为了提高个人的打字速度,一款好的指法练习软件必不可少,本例通过 Visual C++来实现一款指法练习软件的开发。运行本实例,设置练习时间和字符下落间隔,单击“开始”按钮,进行指法练习,在练习时可以在窗体的右侧看到练习的各项数据,程序运行效果如图 3.49 所示。

## 技术要点

在设计指法练习软件时,先在对话框上绘制软件的背景位图,然后动态创建10个静态控件,在用rand函数取出1~26的随机数,根据随机获得的数据判断每个控件显示的字母图片,在定时器中设置控件向下移动,当控件到达下落区域的底部或者用户按下了正在下落的字母时,重新取随机数,并在当前控件上显示图片。再设置一个定时器,在该定时器中根据用户的操作计算正确率、错误数、漏打数等信息,当到达用户设置练习时间后自动停止控件的下落,这样指法练习软件就完成了。

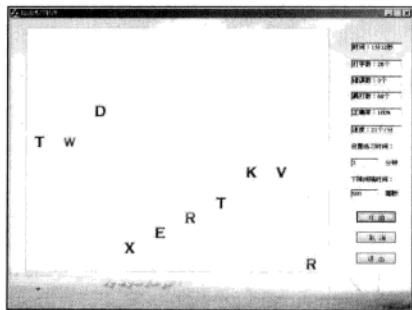


图 3.49 指法练习软件

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加4个静态文本控件、8个编辑框控件和3个按钮控件。
- (3) 向工程导入30个BMP位图资源。
- (4) 在主窗体的头文件中声明变量,代码如下:

```
UINT m_Num[10]; //显示字母
CStatic m_Static[10]; //静态控件
int m_Error; //错误数
int m_Sum; //打字数
int m_Lose; //漏打数
int m_Time; //用时
BOOL m_IsStart; //开始
int m_iTime; //定时
int m_aTime; //间隔时间
```

- (5) 添加 WM\_CTLCOLOR 消息的处理函数,在该函数中调用绘制窗体的背景位图,代码如下:

```
HBRUSH CFingerExerciseDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
 HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
 CBitmap m_BK; //声明位图对象
 m_BK.LoadBitmap(IDB_BITMAPBK); //加载位图资源
 if (nCtlColor == CTLCOLOR_DLG) //如果是对话框
 {
 CBrush m_Brush(&m_BK); //定义一个位图画刷
 CRect rect; //获得窗体客户区域
 GetClientRect(rect);
 pDC->SelectObject(&m_Brush); //选中画刷
 pDC->FillRect(rect,&m_Brush); //填充客户区域
 return m_Brush; //返回画刷
 }
 else
 {
 hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
 if (nCtlColor == CTLCOLOR_STATIC) //如果是静态文本控件
 {
 pDC->SetBkMode(TRANSPARENT); //设置控件背景透明
 }
 return hbr;
 }
}
```

- (6) 添加自定义函数 RandLetter, 该函数用于设置随机数,代码如下:

```
void CFingerExerciseDlg::RandLetter()
{
 CTime time = CTime::GetCurrentTime(); //获得系统时间
 srand(time.GetHour()+time.GetMinute()+time.GetSecond()); //根据时间设置种子
 for(int i=0;i<10;i++)
 {
 m_Num[i] = rand()%26+1; //获得1~26之间的随机数
 }
}
```

- (7) 添加自定义函数 SetBitmap, 该函数用于设置静态控件的显示图片,代码如下:

```
void CFingerExerciseDlg::SetBitmap(int num)
{
 HBITMAP m_hBitmap;
```

```
//加载图片资源
m_hBitmap = LoadBitmap(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDB_BITMAPA+m_Num[num]-1));
m_Static[num].SetBitmap(m_hBitmap); //设置显示图片
m_Static[num].MoveWindow(45+58*num,21,24,24); //移动控件
m_Static[num].ShowWindow(SW_SHOW); //显示控件
}
```

(8) 处理“开始”按钮的单击事件,在该事件的处理函数中清空控件中的数据,并设置定时器,开始练习,代码如下:

```
void CFingerExerciseDlg::OnButstart()
{
 m_Error = 0; //错误数为0
 m_Sum = 0; //打字数为0
 m_Lose = 0; //漏打数为0
 m_Time = 0; //用时为0
 CString time; //获得练习时间
 m_Timing.GetWindowText(time);
 m_iTime = atoi(time)*60; //计算练习时间
 m_Alternation.GetWindowText(time); //获得下降间隔
 m_aTime = atoi(time); //转换下降间隔
 RandLetter(); //设置随机数
 for(int i=0;i<10;i++)
 {
 SetBitmap(i); //设置控件显示图片
 }
 m_IsStart = TRUE; //开始练习
 SetTimer(1,m_aTime,NULL); //设置下降间隔定时器
 SetTimer(2,1000,NULL); //设置练习时间定时器
}
```

(9) 处理主窗体的定时器事件,在该事件的处理函数中设置控件的下落记录漏打数,并设置漏打控件的图片,在练习时间定时器中,判断是否结束练习,如果结束练习则显示用户练习成绩单,并将信息保存到 INI 文件中,代码如下:

```
void CFingerExerciseDlg::OnTimer(UINT nIDEvent)
{
 if(nIDEvent == 1) //下降间隔定时器
 {
 for(int i=0;i<10;i++) //循环10个静态控件
 {
 CRect rect;
 m_Static[i].GetClientRect(rect); //获得控件的客户区域
 m_Static[i].MapWindowPoints(this,rect); //转换坐标
 if(rect.bottom == 495) //如果控件到底底部
 {
 m_Lose++; //记录漏打数
 srand(m_Num[i]*i+m_Num[i]+i); //设置随机种子
 m_Num[i] = rand()%26+1; //获取随机数
 SetBitmap(i); //设置控件显示图片
 rect.top = 21; //设置控件顶部位置
 rect.bottom = 45; //设置控件底部位置
 }
 else //控件未到达底部
 {
 rect.top += 30; //设置控件顶部位置
 rect.bottom += 30; //设置控件底部位置
 }
 m_Static[i].MoveWindow(rect); //移动控件
 }
 }
 else if(nIDEvent == 2) //练习时间定时器
 {
 if(m_Time == m_iTime) //如果练习时间到
 {
 m_IsStart = FALSE; //设置游戏结束
 KillTimer(1); //关闭下降间隔定时器
 KillTimer(2); //关闭练习时间定时器
 CString str;
 //设置成绩单字符串
 str.Format("个 | \r\n | 错误数: %03d个 | \r\n | 漏打数: %03d个 | \r\n | 正确率: %0.0f%% | 速度: %03d个/分 | \r\n | 时间: %02d分%02d秒 | 打字数: %03d",
 m_Num[0], m_Num[1], m_Num[2], m_Num[3], m_Num[4], m_Num[5], m_Num[6], m_Num[7], m_Num[8], m_Num[9],
 m_Error, m_Lose, (m_Sum-m_Error)*1.0/m_Sum*100, m_Sum*60/m_Time);
 MessageBox(str); //显示成绩单
 for(int i=0;i<10;i++)
 m_Static[i].ShowWindow(SW_HIDE); //隐藏静态控件
 }
 }
}
```





```

 }
 else
 {
 m_Time++;
 CString str;
 str.Format("时间: %d分%d秒",m_Time/60,m_Time%60); //格式化时间字符串
 m_uTime.SetWindowText(str); //显示所用时间
 str.Format("打字数: %d个",m_Sum); //格式化打字数
 m_sNum.SetWindowText(str); //显示打字数
 str.Format("错误数: %d个",m_Error); //格式化错误数
 m_eNum.SetWindowText(str); //显示错误数
 str.Format("漏打数: %d个",m_Lose); //格式化漏打数
 m_lNum.SetWindowText(str); //显示漏打数
 str.Format("正确率: %0.0f%%",(m_Sum-m_Error)*1.0/m_Sum*100); //格式化正确率
 m_Veracity.SetWindowText(str); //显示正确率
 str.Format("速度: %d个/分",m_Sum*60/m_Time); //格式化速度
 m_Rate.SetWindowText(str); //显示速度
 }
 CDialog::OnTimer(nIDEvent);
}

```

(10) 重载 PreTranslateMessage 虚函数, 在该虚函数中判断用户按下的键盘键是否对应下落的字母, 如果是, 则重新设置控件的显示图片, 如果不是, 则记录一次错误, 代码如下:

```

BOOL CFingerExerciseDlg::PreTranslateMessage(MSG* pMsg)
{
 if(m_IsStart) //如果开始练习
 {
 if(pMsg->message == WM_KEYDOWN) //按下键盘键
 {
 BOOL IsNum=FALSE; //初始化布尔变量
 for(int i=0;i<26;i++) //循环26次进行判断
 {
 if(pMsg->wParam == 0x0041+i) //判断按下的键值
 {
 for(int j=0;j<10;j++) //循环10个控件
 {
 if(m_Num[j] == i+1) //如果按下的按键是正在下落的字母
 {
 CRect rect;
 m_Static[j].GetClientRect(rect); //获得静态控件的客户区域
 m_Static[j].MapWindowPoints(this,rect); //转换坐标
 srand(m_Num[j]*j*i+m_Num[j]+i); //设置随机数种子
 m_Num[j] = rand()%26+1; //获得随机数
 SetBitmap(j); //重新设置显示的字母
 rect.top = 21; //设置控件顶部位置
 rect.bottom = 45; //设置控件底部位置
 m_Static[j].MoveWindow(rect); //移动控件
 IsNum = TRUE; //设置按下字母正确
 }
 }
 }
 }
 if(!IsNum) //如果按下按键错误
 {
 m_Error++; //错误数加1
 m_Sum++; //打字数加1
 }
 }
 }
 return CDialog::PreTranslateMessage(pMsg);
}

```

(11) 处理“取消”按钮的单击事件, 在该事件的处理函数中关闭定时器并隐藏控件, 代码如下:

```

void CFingerExerciseDlg::OnButcancel()
{
 KillTimer(1); //关闭下降间隔定时器
 KillTimer(2); //关闭练习时间定时器
 for(int i=0;i<10;i++)
 m_Static[i].ShowWindow(SW_HIDE); //隐藏静态控件
 Initialization(); //初始化控件显示
}

```

## 举一反三

根据本实例, 读者可以:

- 实现键盘按键监控。

## 3.10 简单游戏设计

通过前面实例的介绍,相信读者已经对控件和图形图像等知识都有了初步的了解,下面结合前面的内容来设计几款简单的游戏。

## 实例 143 拼图游戏

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\03\143

## 实例说明

本实例实现了拼图游戏,运行程序,在“图像”菜单中可以选择程序中自带的图片或者是用户自己选择的图片,在“级别”菜单中可以选择游戏的级别,在“系统”菜单中可以选择开始游戏,程序运行效果如图 3.50 所示。

## 技术要点

为了制作拼图游戏,首先要动态创建一组静态控件,这些控件将作为拼图块进行移动,用户需要选择拼图图像,可以使用程序中自带的图像,也可以自己选择图片,然后程序将根据用户选择的级别设置静态控件的大小和位置,并将用户选择的图像绘制到静态控件上,当用户选择开始游戏以后,程序将隐藏右下角的拼图块,使拼图空出一个位置,用户可以使用上下左右键来控制剩余拼图块的移动,当用户完成拼图时,游戏提示用户胜利,这样完成了拼图游戏的设计。

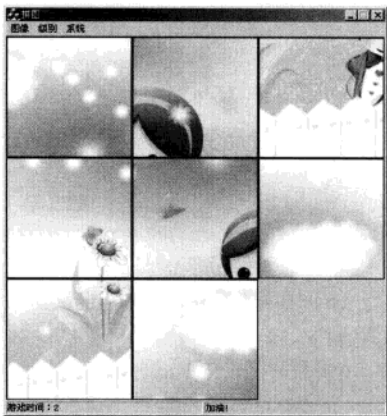


图 3.50 拼图游戏

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向工程中导入一个位图资源,并插入一个菜单资源,为菜单添加相应的子菜单。
- (3) 在主窗口的头文件中声明变量,代码如下:

```
CStatic m_Picture[25]; //控件数组
char m_Path[256]; //程序根目录路径
CString m_pPath; //图片路径
BOOL m_Start; //是否开始游戏
int m_Time; //游戏时间
int m_Size; //控件大小
int m_Num; //级别数
CPoint m_Point[25]; //图片完成位置
CPoint m_MovePoint; //移动块位置
CRect m_Rect[25]; //控件区域
CStatusBar m_Statusbar; //状态栏
CHelpDlg* dlg; //帮助对话框
```

- (4) 在主窗口中初始化时,创建状态栏和静态控件,代码如下:

```
GetCurrentDirectory(256,m_Path); //获取程序根目录路径
m_Start = FALSE;
UINT array[2]={12301,12302};
m_Statusbar.Create(this); //创建状态栏窗口
m_Statusbar.SetIndicators(array,sizeof(array)/sizeof(UINT)); //添加面板
CRect rect;
GetWindowRect(rect); //获得窗口区域
m_Statusbar.SetPanelInfo(0,array[0],rect.Width()/2); //设置面板宽度
m_Statusbar.SetPanelInfo(1,array[1],rect.Width()/2);
CString gTime;
m_Time = 0;
```

```
gTime.Format("游戏时间: %d",m_Time); //设置游戏时间
m_Statusbar.SetPaneText(0,gTime); //设置状态栏显示游戏时间
m_Statusbar.SetPaneText(1,"加油!"); //设置状态栏显示文本
RepositionBars(AFX_IDW_CONTROLBAR_FIRST,AFX_IDW_CONTROLBAR_LAST,0);//显示状态栏
for(int i=0;i<25;i++)
{
 m_Picture[i].Create("",WS_CHILD|WS_CLIPSIBLINGS|WS_EX_TOOLWINDOW|WS_BORDER,
 CRect(0,0,50,50),this,1200+i); //创建静态控件
}
dlg = new CHelpDlg; //声明对话框指针
dlg->Create(IDD_HELP_DIALOG,this); //创建模态对话框
```

(5) 在主窗口中添加自定义函数 SetGrade, 该函数用于根据用户设置的级别判断显示多少个静态控件, 代码如下:

```
void CSpellPictureDlg::SetGrade(int n)
{
 CRect rcdlg;
 GetClientRect(rcdlg); //获得窗口客户区域
 m_Size = rcdlg.Width()/n; //获得每个控件的宽度和高度
 for(int i=0;i<n;i++)
 {
 for(int j=0;j<n;j++)
 {
 CRect rect(0+j*m_Size,0+i*m_Size,m_Size+j*m_Size,m_Size+i*m_Size);//设置控件区域
 m_Picture[j+i*n].MoveWindow(&rect); //移动控件
 m_Picture[j+i*n].ShowWindow(SW_SHOW); //显示区域
 m_MovePoint.x = rect.left; //记录右下角控件的左上角横坐标
 m_MovePoint.y = rect.top; //记录右下角控件的左上角纵坐标
 }
 }
 ShowPicture(n,m_Size); //绘制图片
 for(i=n*n;i<25;i++)
 {
 m_Picture[i].ShowWindow(SW_HIDE); //设置剩余控件隐藏
 m_Picture[i].EnableWindow(FALSE); //设置剩余控件不可用
 }
}
```

(6) 在主窗口中添加自定义函数 ShowPicture, 该函数用于将用户设置的图片绘制到控件上, 代码如下:

```
void CSpellPictureDlg::ShowPicture(int m, int n)
{
 HBITMAP m_hBitmap;
 m_hBitmap = (HBITMAP)::LoadImage(AfxGetInstanceHandle(),m_pPath,
 IMAGE_BITMAP,0,0,LR_LOADFROMFILE|LR_DEFAULTCOLOR|LR_DEFAULTSIZE); //加载位图资源
 int x=0,y=0,i=0,j=0;
 CDC* pDC = m_Picture[0].GetDC(); //控件设备上下文
 CDC memdc;
 memdc.CreateCompatibleDC(pDC); //创建与内存兼容设备上下文
 memdc.SelectObject(m_hBitmap); //将位图选进设备上下文中
 BITMAP bmp;
 GetObject(m_hBitmap,sizeof(bmp),&bmp); //获得图片信息
 x = bmp.bmWidth/m; //设置绘制图片宽度
 y = bmp.bmHeight/m; //设置绘制图片高度
 pDC->StretchBlt(0,0,n,n,&memdc,0,0,x,y,SRCCOPY); //绘制图片
 pDC->Draw3dRect(0,0,n,n,RGB(0,0,0),RGB(0,0,0)); //绘制3D矩形
 UpdateWindow(); //更新窗口显示
 for(i=0;i<m;i++)
 {
 for(j=0;j<m;j++)
 {
 CDC* pDC = m_Picture[j+i*m].GetDC(); //获得控件设备上下文
 pDC->StretchBlt(0,0,n,n,&memdc,x*j*y*i,x,y,SRCCOPY); //绘制图片
 pDC->Draw3dRect(0,0,n,n,RGB(0,0,0),RGB(0,0,0)); //绘制3D矩形
 }
 }
 memdc.DeleteDC();
 UpdateWindow();
}
```

(7) 在主窗口中添加自定义函数 RandPlace, 该函数用于随机变化控件的显示位置, 代码如下:

```
void CSpellPictureDlg::RandPlace(int a, int b)
{
 int num[25];
 int n,m=0,k=1;
 CTime time = CTime::GetCurrentTime(); //获得系统时间
 srand(time.GetSecond()); //设置随机数种子
```

```

num[0] = rand()%(a*a-1); //求随机数
while(k != (a*a-1)) //根据获得随机数的数量循环
{
 n = rand()%(a*a-1); //获得随机数
 for(int i=0;i<k;i++)
 {
 if(num[i] == n) //如果随机数存在
 {
 m = 1; //标记m为1
 }
 }
 if(m == 0) //如果随机数不存在
 {
 k++; //随机数数量加1
 num[k-1] = n; //保存随机数
 }
 m = 0; //标记随机数不存在
}
for(int i=0;i<a;i++)
{
 for(int j=0;j<a;j++)
 {
 m_Point[j+i*a].x = j*b; //记录控件左上角横坐标
 m_Point[j+i*a].y = i*b; //记录控件左上角纵坐标
 }
}
for(i=0;i<(a*a-1);i++)
{
 int p;
 p = num[i]; //获得随机数
 m_Rect[i].left = m_Point[p].x; //记录随机显示后的控件左边界
 m_Rect[i].top = m_Point[p].y; //记录随机显示后的控件上边界
 m_Rect[i].right = m_Rect[i].left + b; //记录随机显示后的控件右边界
 m_Rect[i].bottom = m_Rect[i].top + b; //记录随机显示后的控件下边界
 m_Picture[i].MoveWindow(&m_Rect[i]); //移动控件位置
}

```

(8) 处理“开始游戏”菜单项的单击事件,在该事件的处理函数中设置定时器,调用 RandPlace 函数变化控件位置,并开始游戏,代码如下:

```

void CSpellPictureDlg::OnMenustart()
{
 if(m_pPath.IsEmpty()) //图片路径不为空
 {
 MessageBox("请选择图片!");
 return;
 }
 m_Time = 0; //设置用时
 SetTimer(1,1000,NULL); //设置定时器
 m_Start = TRUE; //开始游戏
 CRect rcdlg;
 GetClientRect(rcdlg); //获得窗口客户区域
 m_Size = rcdlg.Width()/m_Num; //设置图像的大小
 RandPlace(m_Num,m_Size); //随机变化控件位置
 m_Picture[m_Num*m_Num-1].ShowWindow(SW_HIDE); //隐藏右下角的控件
}

```

(9) 处理主窗口的定时器事件,在该事件的处理函数中设置游戏进行时间,代码如下:

```

void CSpellPictureDlg::OnTimer(UINT nIDEvent)
{
 m_Time++; //用时加1
 CString gTime;
 gTime.Format("游戏时间: %d",m_Time); //格式化游戏时间
 m_Statusbar.SetPaneText(0,gTime); //显示游戏时间
 CDialog::OnTimer(nIDEvent);
}

```

(10) 在主窗口中添加自定义函数 GameWin,该函数用于判断用户是否完成拼图,如果完成,则提示用户胜利,代码如下:

```

void CSpellPictureDlg::GameWin()
{
 int num=0;
 for(int i=0;i<m_Num*m_Num-1;i++)
 {
 if(m_Point[i].x == m_Rect[i].left && m_Point[i].y == m_Rect[i].top) //如果控件位置等于变化前的位置
 num++; //标记相同数变量加1
 }
 if(num == m_Num*m_Num-1) //如果所有控件位置都等于变化前的位置

```



```

 {
 KillTimer(1);
 m_Start = FALSE;
 m_Picture[num].ShowWindow(SW_SHOW);
 MessageBox("获胜了, 你好棒啊!");
 }
}
//停止定时器
//标记游戏结束
//显示右下角的控件
//提示用户胜利

```

(11) 重载主窗口的 PreTranslateMessage 虚函数, 在该虚函数中设置用户按下上下左右键时移动控件, 代码如下:

```

BOOL CSpellPictureDlg::PreTranslateMessage(MSG* pMsg)
{
 if(m_Start)
 {
 BOOL move = FALSE;
 if(pMsg->message == WM_KEYDOWN && pMsg->wParam == 0x0026) //按下上箭头
 {
 for(int i=0;i<9;i++)
 {
 if(m_Rect[i].left == m_MovePoint.x &&
 m_Rect[i].top-m_Size == m_MovePoint.y && !move) //如果控件上方是空位
 {
 CPoint point;
 point = m_MovePoint;
 m_Picture[i].MoveWindow(m_MovePoint.x,m_MovePoint.y,m_Size,m_Size); //获得空位左上角坐标
 //移动控件
 m_MovePoint.x = m_Rect[i].left;
 m_MovePoint.y = m_Rect[i].top;
 //将控件位置设为空位
 m_Rect[i].left = point.x;
 m_Rect[i].top = point.y;
 //记录控件的左边界
 m_Rect[i].right = m_Rect[i].left+point.x;
 //记录控件的右边界
 m_Rect[i].bottom = m_Rect[i].top+point.y;
 //记录控件的下边界
 move = TRUE;
 }
 }
 }
 else if(pMsg->message == WM_KEYDOWN && pMsg->wParam == 0x0028) //按下下箭头
 {
 for(int i=0;i<9;i++)
 {
 if(m_Rect[i].left == m_MovePoint.x &&
 m_Rect[i].top+m_Size == m_MovePoint.y && !move) //如果控件下方是空位
 {
 CPoint point;
 point = m_MovePoint;
 m_Picture[i].MoveWindow(m_MovePoint.x,m_MovePoint.y,m_Size,m_Size); //获得空位左上角坐标
 //移动控件
 m_MovePoint.x = m_Rect[i].left;
 m_MovePoint.y = m_Rect[i].top;
 //将控件位置设为空位
 m_Rect[i].left = point.x;
 m_Rect[i].top = point.y;
 //记录控件的左边界
 m_Rect[i].right = m_Rect[i].left+point.x;
 //记录控件的右边界
 m_Rect[i].bottom = m_Rect[i].top+point.y;
 //记录控件的下边界
 move = TRUE;
 }
 }
 }
 else if(pMsg->message == WM_KEYDOWN && pMsg->wParam == 0x0025) //按下左箭头
 {
 for(int i=0;i<9;i++)
 {
 if(m_Rect[i].left-m_Size == m_MovePoint.x &&
 m_Rect[i].top == m_MovePoint.y && !move) //如果控件左方是空位
 {
 CPoint point;
 point = m_MovePoint;
 m_Picture[i].MoveWindow(m_MovePoint.x,m_MovePoint.y,m_Size,m_Size); //获得空位左上角坐标
 //移动控件
 m_MovePoint.x = m_Rect[i].left;
 m_MovePoint.y = m_Rect[i].top;
 //将控件位置设为空位
 m_Rect[i].left = point.x;
 m_Rect[i].top = point.y;
 //记录控件的左边界
 m_Rect[i].right = m_Rect[i].left+point.x;
 //记录控件的右边界
 m_Rect[i].bottom = m_Rect[i].top+point.y;
 //记录控件的下边界
 move = TRUE;
 }
 }
 }
 else if(pMsg->message == WM_KEYDOWN && pMsg->wParam == 0x0027) //按下右箭头
 {
 for(int i=0;i<9;i++)
 {

```

```

if(m_Rect[i].left+m_Size == m_MovePoint.x &&
 m_Rect[i].top == m_MovePoint.y && !move) //如果控件右方是空位
{
 CPoint point;
 point = m_MovePoint;
 m_Picture[i].MoveWindow(m_MovePoint.x,m_MovePoint.y,m_Size,m_Size);//移动控件
 m_MovePoint.x = m_Rect[i].left; //将控件位置设为空位
 m_MovePoint.y = m_Rect[i].top;
 m_Rect[i].left = point.x; //记录控件的左边界
 m_Rect[i].top = point.y; //记录控件的上边界
 m_Rect[i].right = m_Rect[i].left+point.x; //记录控件的右边界
 m_Rect[i].bottom = m_Rect[i].top+point.y; //记录控件的下边界
 move = TRUE;
}
}
move = FALSE;
GameWin(); //判断是否获胜
}
return CDialog::PreTranslateMessage(pMsg);
}

```

### 举一反三

根据本实例，读者可以：

- 开发类似华容道的拼图游戏。

## 实例 144 黑白棋

本实例可以提高基础技能

实例位置：光盘\mingrisoft\03\144

### 实例说明

黑白棋又称翻转棋，是一款非常受用户欢迎的棋牌类游戏，可以分为人机对战和人人对战两种，本例通过 Visual C++设计一款人人对战的黑白棋游戏。运行本实例，在黑白棋服务器端单击“服务器设置”按钮设置服务器，然后在黑白棋客户端单击“开始游戏”按钮连接服务器并开始游戏，效果如图 3.51 和图 3.52 所示。

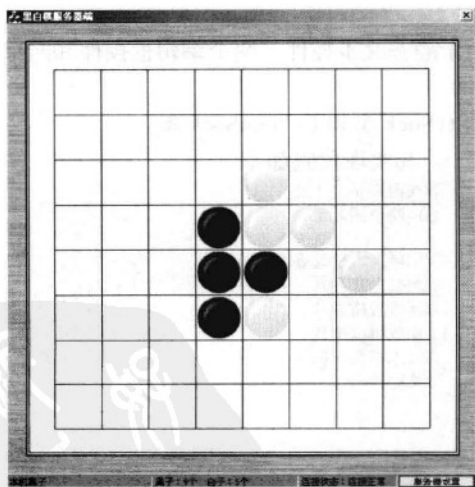


图 3.51 黑白棋服务器端

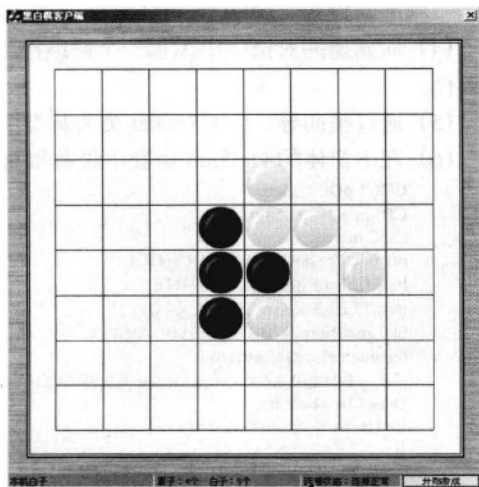


图 3.52 黑白棋客户端

### 技术要点

在设计黑白棋游戏时，由于是人和人之间的对战，所以要使用套接字设计网络连接，在对



话框上绘制软件的背景位图,在背景中绘制纵、横各8个方格,黑白棋的棋子是下在方格中的,在落子时,两个相同颜色棋子(己方)之间的另一种颜色棋子(对方)将被转换为相同颜色(己方),但是用户落子的位置需要注意,只有能将对方棋子转换为自己棋子的位置才可以落子,如果没有符合条件的落子位置,对方将连续落子,当棋盘中没有空格或者双方都不可以落子时,判断游戏结束,棋子多的一方将获得胜利。

本示例中设计黑白棋游戏时,主要用 Send 方法和 Receive 方法来进行数据的发送和接收,下面对本实例中用到的关键技术进行详细讲解。

#### (1) Send 方法

Send 方法用于发送数据到连接的套接字上,其语法格式如下:

```
virtual int Send(const void* lpBuf, int nBufLen, int nFlags = 0);
```

参数说明:

- lpBuf: 标识要发送数据的缓冲区。
- nBufLen: 确定缓冲区的大小。
- nFlags: 标识函数调用模式。

#### (2) Receive 方法

Receive 方法用于从一个套接字上接收数据,其语法格式如下:

```
virtual int Receive(void* lpBuf, int nBufLen, int nFlags = 0);
```

参数说明:

- lpBuf: 是接收数据的缓冲区。
- nBufLen: 确定缓冲区的长度。
- nFlags: 确定函数的调用模式。

### 实现过程

- (1) 新建一个基于对话框的应用程序,将其窗体标题改为“黑白棋服务器端”。
- (2) 向工程导入3个BMP位图资源,用来绘制棋盘和棋子,向对话框中添加一个按钮控件。
- (3) 创建一个新的对话框资源,修改其ID为IDD\_SETSERVER\_DIALOG,将其窗体标题改为“服务器设置”,并设置对话框显示的字体信息。
- (4) 向新建的对话框中添加一个群组控件、两个静态文本控件、两个编辑框控件和两个按钮控件。
- (5) 通过类向导,以CSocket类为基类派生CSrvSock类和CClientSock类。
- (6) 在主窗体的OnPaint函数中绘制棋盘和棋子,其实现代码如下:

```
CDC* pDC = GetDC(); //获得设备上下文
CBitmap bmp1,bmp2,bk; //声明位图对象
CDC memdc;
memdc.CreateCompatibleDC(pDC); //创建兼容的设备上下文
bmp1.LoadBitmap(IDB_WHITE); //加载白棋图片
bmp2.LoadBitmap(IDB_BLACK); //加载黑棋图片
bk.LoadBitmap(IDB_CHESSBOARD); //加载棋盘图片
memdc.SelectObject(&bk); //选入棋盘对象
pDC->BitBlt(0,0,600,600,&memdc,0,0,SRCCOPY); //绘制棋盘
DrawChessboard();
for (int m=0; m<row-1; m++)
{
 for (int n=0; n<col-1; n++)
 {
 if (m_NodeList[m][n].m_Color == ncWHITE) //如果当前节点是白子
 {
 memdc.SelectObject(&bmp1); //选入白子对象
 pDC->BitBlt(m_NodeList[m][n].m_Rect.left+3,m_NodeList[m][n].m_Rect.top+3,
 55,55,&memdc,0,0,SRCCOPY); //绘制白子
```

```

 }
 else if(m_NodeList[m][n].m_Color == ncBLACK)//如果当前节点是黑子
 {
 memdc.SelectObject(&bmp2); //选入黑子对象
 pDC->BitBlt(m_NodeList[m][n].m_Rect.left+3,m_NodeList[m][n].m_Rect.top+3,
 55,55,&memdc,0,0,SRCCOPY); //绘制黑子
 }
}
bk.DeleteObject();
ReleaseDC(&memdc);

```

### 举一反三

根据本实例，读者可以：

- 设计网络跳棋、五子棋游戏。

## 实例 145 俄罗斯方块

本实例可以提高基础技能

实例位置：光盘\mingrisoft\03\145

### 实例说明

相信许多人都玩过俄罗斯方块游戏，该游戏即可以锻炼反应能力，也可以提高思维能力。笔者小时候就为该游戏所痴迷，就梦想长大后也设计一个类似的游戏。如今选择了程序员的职业，自然可以实现这个“梦想”了。本程序设计效果如图 3.53 所示，运行效果如图 3.54 所示。

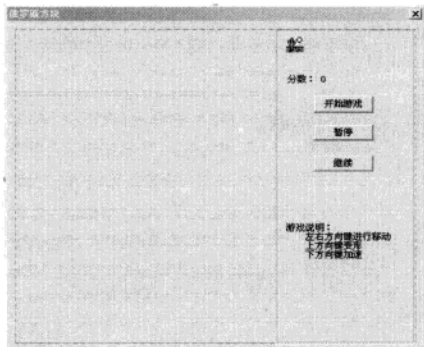


图 3.53 俄罗斯方块设计图

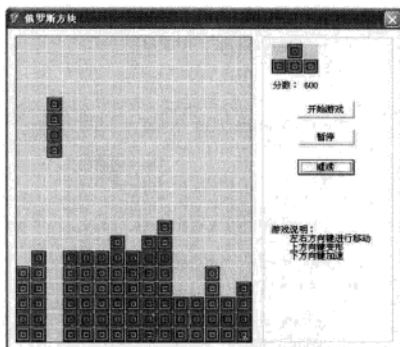


图 3.54 俄罗斯方块运行图

### 技术要点

在实现俄罗斯方块游戏时，涉及的技术比较简单，主要有两个，一是在绘制表格时使用内存画布来绘制表格，减少屏幕刷新次数，二是随机产生不同类型的方块。首先将所有的绘图操作在内存画布上完成，然后在内存画布对象释放时，将其内容绘制到窗口中，这样在窗口中只进行了一次绘图，减少了屏幕的刷新次数；其次在游戏过程中，需要产生不同类型的方块，为此需要编写一个方法，来生成方块。由于方块的基本类型只有 7 种，因此产生的类型必须在 1 到 7 之间。为此，需要使用随机函数 rand 与 7 进行取模运算。

在设计俄罗斯方块时，首先需要设计游戏的表格（CGrid 类），游戏中的表格由单元格（CCell 类）构成。在单元格对象中描述了所在表格的行和列索引，单元格是否被使用、单元格是否被固定（当图像移动到表格的底部，不能再向下移动了）等信息。

俄罗斯方块中共有 7 种基本形状，基本类型可以按 90° 旋转，这样又可以变换出 12 种新的



类型, 为这些形状统一建一个类 CPiece。在 CPiece 类中, 有一个关键的成员 m\_ImagesPT, 该成员是一个数组, 其中包含了 4 个 CPoint 对象。由于俄罗斯方块图像都占有 4 个单元格, 因此使用 4 个 CPoint 对象描述每一个单元格坐标, 这里的坐标是单元格的行和列索引。当游戏产生一个新的方块图像时, 图像最左边的位置由 m\_nLeftIndex 成员表示, 顶部位置由 m\_nTopIndex 成员表示。

游戏中显示方块时, 依据这两个成员和当前图像的类型将遍历 CPiece 的 m\_ImagesPT 成员, 读取每个元素表示的单元格坐标, 通过该坐标来定位表格中的单元格 CCell, 将单元格 CCell 的 m\_bUsed 成员设置为 TRUE, 然后调用表格 CGrid 的 DrawGrid 来绘制表格, 使得方块显示在表格中。

在游戏进行中, 默认方块会向下移动, 用户也可以使用左右方向键来移动图像。为了能够移动图像, 需要设置 CPiece 类的 m\_nLeftIndex 成员和 m\_nTopIndex 成员, 然后重新设置 CPiece 类的 m\_ImagesPT 成员, 最后绘制表格就可以实现移动图像的效果了。

## 实现过程

- (1) 创建一个基于对话框的工程, 工程名称为 “RussianGrid”。
- (2) 向对话框中添加图片、按钮、静态文本和群组框的控件。
- (3) 向对话框中添加 HorMovePiece 方法, 当用户按左右方向键时左右移动方块图像。其实现代码如下:

```
void CRussianGridDlg::HorMovePiece(int nOffset)
{
 if (m_Piece.m_bMoving == FALSE) //方块是否处于移动过程中
 {
 return;
 }
 BOOL bLeftMove = (nOffset < 1)? TRUE: FALSE; //防止左右穿透图像
 if (!m_pGrid->IsAllowHorMove(m_Piece, bLeftMove))
 {
 m_bKeyDown = FALSE; //如果不能左右移动, 则在OnTimer中继续向下移动
 return;
 }

 m_Piece.MovePiece(m_Piece.m_nLeftIndex + nOffset, m_Piece.m_nTopIndex);
 for(int i=0; i<4; i++) //显示图像
 {
 int nX = m_Piece.m_ImagesPT[i].x;
 int nY = m_Piece.m_ImagesPT[i].y;
 m_pGrid->m_CellGrid[nY][nX].m_bUsed = TRUE;
 }
 m_pGrid->DrawGrid(GetDC()); //绘制表格
 for(i=0; i<4; i++) //在绘制下一次图像时使之前的图像消失
 {
 int nX = m_Piece.m_ImagesPT[i].x;
 int nY = m_Piece.m_ImagesPT[i].y;
 m_pGrid->m_CellGrid[nY][nX].m_bUsed = FALSE;
 }
 //获取当前方块所在列对应的底部行索引
 BOOL bRet = m_pGrid->GetBottomRowIndex(m_Piece.m_nLeftIndex, m_Piece.m_nWidth, m_Piece);
 if (bRet)
 {
 for(i=0; i<4; i++)
 {
 int nX = m_Piece.m_ImagesPT[i].x;
 int nY = m_Piece.m_ImagesPT[i].y;
 m_pGrid->m_CellGrid[nY][nX].m_bUsed = FALSE;
 if (m_nKeyDown > 3)
 {
 m_pGrid->m_CellGrid[nY][nX].m_bFixed = TRUE;
 m_Piece.m_bMoving = FALSE;
 GradeHandle(m_Piece); //计算成绩
 if (GameOver()) //判断游戏是否结束
 {
 m_bGameRunning = FALSE;
 MessageBox("游戏结束!");
 }
 }
 }
 }
}
```

## 举一反三

根据本实例，读者可以：

- 设计贪食蛇游戏。

## 实例 146 快来打地鼠

本实例可以提高基础技能

实例位置：光盘\mingrisoft\03\146

## 实例说明

本实例实现的是在 5 个洞口随机出现可爱的地鼠，当用鼠标点击到地鼠后，地鼠会伸长舌头，表明你已经打到它了，效果如图 3.55 所示。

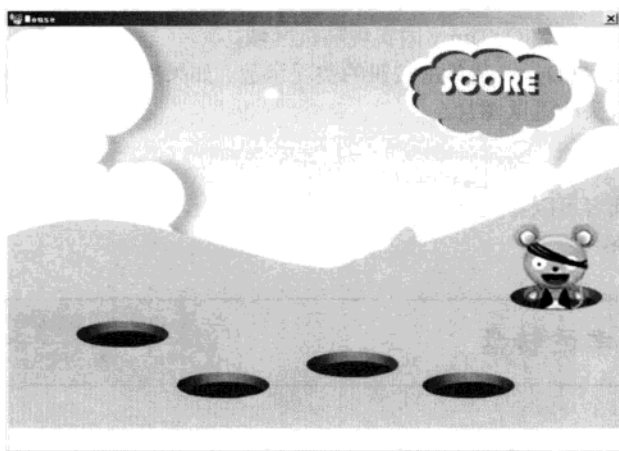


图 3.55 可以选择播放曲目的 CD 播放器

## 技术要点

实例中一共有 5 个地鼠洞，每个地鼠洞都是一个矩形区域，在定时器内随机获取 5 个区域中的一个，然后在该区域内绘制地鼠图像，如果用户在该区域内按下了鼠标，表明已经打到地鼠，需要在该区域内重新绘制老鼠图片，绘制伸长舌头地鼠图像。本实例是在 OnPaint 方法内通过一个图像编号来绘制地鼠图像，图像编号是多少就绘制该编号的图像，其他编号的图像不被显示，也就实现了隐藏。

## 实现过程

- (1) 创建基于对话框的工程，将对话框的 ID 设置为 IDD\_MOUSE\_DIALOG。
- (2) 向工程中添加光标资源，将光标资源的 ID 设置为 IDC\_BROWSE。
- (3) 在 OnInitDialog 函数中设置区域坐标，并设置定时器。
- (4) 在函数 OnPaint 中绘制地鼠图片，如果随机区域号变量 m\_iCurRand 值为-1，就不绘制任何地鼠图片。

```
void CMouseDlg::OnPaint()
{
 if (IsIconic())
 {
 CPaintDC dc(this);
```

```

//代码省略
}
else
{
 CDialog::OnPaint();
}
if (m_iCurRand > -1)
{
 CDC* pDC = GetDC(); //获取设备上下文指针
 CBitmap bmp;
 bmp.LoadBitmap(IDB_MOUSE); //加载图片
 CDC memDC;
 memDC.CreateCompatibleDC(pDC); //创建兼容设备上下文
 memDC.SelectObject(&bmp); //加载图片到设备上下文
 pDC->BitBlt(m_rcCur.left, m_rcCur.top, m_rcCur.Width(),
 m_rcCur.Height(), &memDC, 0, 0, SRCCOPY); //绘制图片
 bmp.DeleteObject();
 memDC.DeleteDC();
}
}

```

(5) 在定时器实现函数 OnTimer 内实现随机区域。

(6) 函数 OnLButtonDown 是单击按钮的实现函数, 如果用户点击到地鼠, 就重新绘制地鼠的图片, 绘制伸长舌头的地鼠图片。

### 举一反三

根据本实例, 读者可以:

- 设计 20 点游戏。

## 实例 147 幸运转盘

本实例可以提高基础技能

实例位置: 光盘\mingrisoft\03\147

### 实例说明

幸运转盘是一款转盘抽奖游戏, 单击转盘中央的“开始”按钮后, 转盘开始转动, 转盘停止后, 指针会指向不同的奖品, 效果如图 3.56 所示。

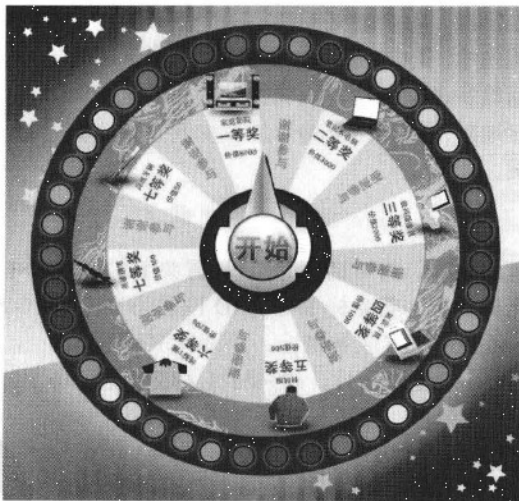


图3.56 幸运转盘

## 技术要点

程序可以使用多个图片来完成，最底层为背景图，中间层转盘图，最上边为指针图；只要使转盘图按照中心点旋转就可以实现最终的效果。图片可以使用 PNG 格式的图片，然后使用 GDI+库来实现图片的旋转。

程序的难点体现在如何使用 GDI+库来实现图片的旋转。PNG 图片可以先转化为流，然后通过 GDI+库的 Image 类打开，绘制的时候需要使用 Graphics 类的 DrawImage 函数，旋转需要使用 Matrix 类来实现，GDI+中有两个旋转函数 Rotate 和 RotateAt，两者的区别是后者可以指定旋转点，用 Matrix 类来实现旋转，其实质就是进行坐标的转换。

坐标的转换的过程是首先定义单位矩阵，将原点设置为 (0,0)，然后在单位矩阵上运行旋转函数，旋转函数通过旋转角度可以改变单位矩阵的值，最后通过坐标变换函数，就可以实现将指定点转换为旋转后的点。

注意：GDI+库有独立的头文件以及链接库，使用时应设置 Visual C++的搜索路径 (Tools/Options/Directories) 使编译器能够找到头文件及链接库。

## 实现过程

(1) 创建对话框应用程序。

(2) 将 PNG 图片添加到工程内，并建立 PNG 资源组，将背景图的 ID 设置为 IDR\_BK，将转盘图的 ID 设置为 IDR\_DISK，将开始按钮图片的 ID 属性设置为 IDR\_START，将按下开始按钮后的图片 ID 属性设置为 DR\_STOP。

(3) 在 StdAfx.h 文件中添加 gdiplus.h 头文件和 gdiplus.lib 库文件的引用。

(4) 方法 OnInitDialog 是对话框初始化的实现，首先从动态链接库 User32 中获取 UpdateLayered Window 函数，然后调用 ImageFromIDResource 函数创建 PNG 图片的 Image 指针，最后通过 DrawDisk 函数将图片绘制出来。

```

BOOL CDiskDlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 //代码省略
 this->MoveWindow(0,0,700,700); //设置窗体宽和高
 GdiplusStartup(&m_pGdiToken,&m_Gdiplus,NULL); //创建GDI+句柄
 m_hInstance = LoadLibrary("User32.DLL"); //加载User32.DLL动态链接库
 if(m_hInstance) //使指针指向链接库中UpdateLayeredWindow函数
 UpdateLayeredWindow=(MYFUNC)GetProcAddress(m_hInstance,
 "UpdateLayeredWindow");
 else
 {
 MessageBox("连接库加载失败","提示",MB_OK);
 exit(0);
 }

 time_t t; //声明时间结构变量
 m_iRand = time(&t); //获取时间，用于随机种子
 m_Blend.BlendOp=0; //设置操作系统
 m_Blend.BlendFlags=0; //属性标识设置
 m_Blend.AlphaFormat=1; //设置Alpha格式
 m_Blend.SourceConstantAlpha=255; //设置Alpha颜色
 ImageFromIDResource(IDR_DISK,"PNG",m_pImageDisk); //创建转盘图片流
 ImageFromIDResource(IDR_BK,"PNG",m_pImageBk); //创建背景图片流
 ImageFromIDResource(IDR_START,"PNG",m_pImageStart); //创建开始按钮图片流
 ImageFromIDResource(IDR_STOP,"PNG",m_pImageStop); //创建按下的开始按钮图片流
 m_StartWidth =m_pImageStart->GetWidth(); //获取开始按钮图片的宽度
 m_StartHeight =m_pImageStart->GetHeight(); //获取开始按钮图片的高度
 DrawDisk(); //绘制图片
 return TRUE;
}

```



(5) 方法 DrawDisk 是绘制图片的实现。首先创建一个兼容的设备上下文, 并加载一个内存图, 然后将 Graphics 对象绑定到设备上下文上, 定义绘制图片所使用的 3 个点的坐标。然后通过 DrawImage 将图片绘制出来, 需要旋转的图片需要定义一个单位矩阵, 单位矩阵需要 3 个点的坐标, 分别是左上、右下、中心点, 使用 RotateAt 函数旋转完矩阵后, 就可以通过 TransformPoints 来改变指定的坐标。最后仍然通过 DrawImage 函数将图片绘制出来。使用 GDI+ 来显示 PNG 图片需要使用窗体的特殊层, 通过函数 UpdateLayeredWindow 可以使用特殊层, 并通过 GetWindowLong 函数修改窗体原有属性。

(6) 自定义方法 ImageFromIDResource 实现将资源内的图片转换为流。首先使用 FindResource 找到资源, 然后将资源的内容读取到由 GlobalAlloc 分配的缓存中, 最后根据 HGLOBAL 对象生成流。

### 举一反三

根据本实例, 读者可以:

- 在窗体背景上绘制图片。

## 3.11 OpenGL 程序设计

OpenGL 是开发三维立体动画的主要函数库, 通过该库很容易实现物体的三维立体效果, 本节将通过几个实例介绍如何通过 OpenGL 开发应用程序。

### 实例 148 制作 OpenGL 动画

这是一个自娱自乐的实例

实例位置: 光盘\mingrisoft\03\148

#### 实例说明

本实例通过对 OpenGL 库的引用实现一个带纹理的茶壶绕水平中心线旋转的动画, 程序启动后显示的是一个静态的茶壶, 如图 3.57 所示, 实现动画效果需要选择“查看”/“开始动画”菜单命令。

#### 技术要点

在 MFC 的视图中绘制 OpenGL 需要将窗体样式设置为 WS\_CLIPSIBLINGS 和 WS\_CLIPCHILDREN。绘制 OpenGL 动画的过程是: 通过 ChoosePixelFormat 函数选择像素格式, SetPixelFormat 函数设置像素格式, wglCreateContext 函数创建环境设备, wglMakeCurrent 函数设置环境设备, glViewport 函数设置浏览视图的大小, glMatrixMode 函数设置矩阵模式, gluPerspective 函数设置透视坐标, glRotatef 函数设置观看坐标, 然后使用 glPushMatrix 函数将数据入栈, glPopMatrix 函数将数据出栈, SwapBuffers 函数交换缓存内容, 最后将要绘制的图像放在 glPushMatrix 函数和 glPopMatrix 函数之间即可。

程序中直接使用 auxSolidTeapot 函数来绘制一个茶壶。通过 glRotatef 函数不断变换视点来达到旋转的动画效果。

#### 实现过程

- (1) 新建一个基于单文档的应用程序。

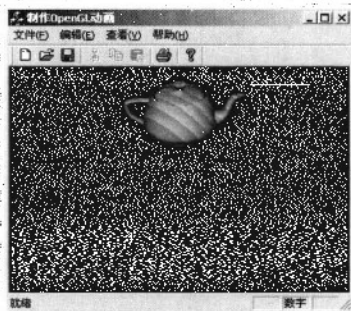


图 3.57 制作 OpenGL 动画

(2) 在工程中加入对 opengl32.lib、glu32.lib、glaux.lib 库的引用。

(3) 新建菜单项, 设置 ID 属性为 ID\_ACT, Caption 属性为“开始动画”, 通过类向导添加菜单响应函数 OnAct。

(4) 添加 OpenGL 头文件的引用, 代码如下:

```
#include "gl\gl.h"
#include "gl\glu.h"
#include "gl\glaux.h"
```

(5) 在实现文件 OPENGLActView.cpp 中加入如下变量声明:

```
GLfloat sgenparams[] = {1.0, 1.0, 1.0, 0.0};
GLubyte stripeImage[3*64];
```

(6) 在 PreCreateWindow 函数中设置窗体样式, 代码如下:

```
BOOL COPENGLActView::PreCreateWindow(CREATESTRUCT& cs)
{
 cs.style |= WS_CLIPSIBLINGS | WS_CLIPCHILDREN; //设置样式
 return CView::PreCreateWindow(cs);
}
```

(7) 在 OnDraw 函数中实现绘制, 代码如下:

```
void COPENGLActView::OnDraw(CDC* pDC)
{
 COPENGLActDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 static GLfloat xangle = 10.0f;
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glPushMatrix ();
 glTranslatef(0.0f, 0.0f, -8.5f);
 glRotatef(xangle, 1.0f, 0.0f, 0.0f);
 xangle += 10.0f;
 auxSolidTeapot(1.5);
 glPopMatrix ();
 glFlush();
 if(FALSE==::SwapBuffers(m_pContextDC->GetSafeHdc()))
 AfxMessageBox("交换缓冲区失败");
}
```

(8) 在 OnCreate 函数中实现像素的设置、设备环境和三维设置方面的初始化, 代码如下:

```
int COPENGLActView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
 if (CView::OnCreate(lpCreateStruct) == -1)
 return -1;
 m_pContextDC=new CClientDC(this);
 ASSERT(m_pContextDC);
 static PIXELFORMATDESCRIPTOR pfd =
 {sizeof(PIXELFORMATDESCRIPTOR),1,
 PFD_DRAW_TO_WINDOW|PFD_SUPPORT_OPENGL|PFD_DOUBLEBUFFER,PFD_TYPE_RGBA,
 24, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 16, 0,
 0, PFD_MAIN_PLANE, 0, 0, 0, 0 };
 int pixelformat=::ChoosePixelFormat(m_pContextDC->GetSafeHdc(),&pfd);
 if(pixelformat==0)
 {
 AfxMessageBox("初始化失败");
 return -1;
 }
 SetPixelFormat(m_pContextDC->GetSafeHdc(),pixelformat,&pfd);
 m_hOpenglRC=wglCreateContext(m_pContextDC->GetSafeHdc());
 wglMakeCurrent(m_pContextDC->GetSafeHdc(),m_hOpenglRC);
 glClearColor(0.0, 0.0, 0.0, 0.0);
 int j;
 for (j = 0; j < 64; j++)
 {
 stripeImage[3*j] = 255;
 stripeImage[3*j+1] =255-2*j;
 stripeImage[3*j+2] =255;
 }
}
```

```
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage1D(GL_TEXTURE_1D, 0, 3, 64, 0, GL_RGB, GL_UNSIGNED_BYTE, stripeImage);
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_S, GL_OBJECT_PLANE, sgenparams);
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_1D);
glEnable(GL_CULL_FACE);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_AUTO_NORMAL);
glEnable(GL_NORMALIZE);
glFrontFace(GL_CW);
glCullFace(GL_BACK);
glMaterialf(GL_FRONT, GL_SHININESS, 64.0);
return 0;
}
```

(9) 关闭窗口后关闭设备环境句柄, 代码如下:

```
BOOL COPENGLActView::Destroy Window()
{
 wglDeleteContext(m_hOpenglRC);
 delete m_pContextDC;
 return CView::Destroy Window();
}
```

(10) 在 OnSize 函数中进行观看角度和视图大小等方面的设置, 代码如下:

```
void COPENGLActView::OnSize(UINT nType, int cx, int cy)
{
 if (0 >= cx || 0 >= cy)
 {
 return;
 }
 CView::OnSize(nType, cx, cy);
 ::glViewport(0, 0, cx, cy);
 ::glMatrixMode(GL_PROJECTION);
 ::glLoadIdentity();
 GLdouble dblaspect = (GLdouble)cx / (GLdouble)cy;
 ::gluPerspective(40.0f, dblaspect, 0.1f, 20.0f);
 ::glMatrixMode(GL_MODELVIEW);
 ::glLoadIdentity();
 ::glTranslatef(0.0f, 0.0f, -5.0f);
 ::glRotatef(20.0f, 1.0f, 0.0f, 0.0f);
}
```

(11) 利用定时器实现动画效果, 代码如下:

```
void COPENGLActView::OnTimer(UINT nIDEvent)
{
 this->OnDraw(this->GetDC()); //绘制动画
 CView::OnTimer(nIDEvent);
}
```

(12) “查看” / “开始动画” 菜单的实现函数如下:

```
void COPENGLActView::OnAct()
{
 SetTimer(1, 15, NULL); //设置定时器
}
```

## 举一反三

根据本实例, 读者可以:

- 利用 OpenGE 绘制位图变换。

## 实例 149

## 利用 OpenGL 绘制立体模型

这是一个自娱自乐的实例

实例位置: 光盘\mingrisoft\03\149

## 实例说明

本实例通过对 OpenGL 库的引用, 实现了绘制正方体的功能, 如图 3.58 所示, 并且正方体每个顶点的颜色各不相同。

## 技术要点

绘制立方体需要在两个矩阵函数 `glPushMatrix` 和 `glPopMatrix` 之间设置立方体顶点的数据, 然后将 `glBegin` 函数的参数设置为 `GL_QUADS`。函数 `glNormal3fv` 用于设置当前点, 函数 `glVertex3fv` 用于设置立方体顶点, 函数 `glColor3fv` 用于设置顶点的颜色值。

## 实现过程

- (1) 新建一个基于单文档的应用程序。
- (2) 在工程中加入对 `opengl32.lib`、`glu32.lib`、`glaux.lib` 库的引用。
- (3) 添加 OpenGL 头文件的引用, 代码如下:

```
#include "gl.h"
#include "glu.h"
#include "gl\glaux.h"
class COPENGLCubeView : public CView
{
protected:
//加入变量的声明
CDC* m_pContextDC;
HGLRC m_hOpenGLRC;
CRect oldrc;
}
//对正方体的各个顶点及顶点的颜色进行初始化
static GLfloat p1[]={0.5,-0.5,-0.5};
static GLfloat p2[]={0.5,0.5,-0.5};
static GLfloat p3[]={0.5,0.5,0.5};
static GLfloat p4[]={0.5,-0.5,0.5};
static GLfloat p5[]={-0.5,-0.5,0.5};
static GLfloat p6[]={-0.5,0.5,0.5};
static GLfloat p7[]={-0.5,0.5,-0.5};
static GLfloat p8[]={-0.5,-0.5,-0.5};

static GLfloat m1[]={1.0,0.0,0.0};
static GLfloat m2[]={-1.0,0.0,0.0};
static GLfloat m3[]={0.0,1.0,0.0};
static GLfloat m4[]={0.0,-1.0,0.0};
static GLfloat m5[]={0.0,0.0,1.0};
static GLfloat m6[]={0.0,0.0,-1.0};

static GLfloat c1[]={0.0,0.0,1.0};
static GLfloat c2[]={0.0,1.0,1.0};
static GLfloat c3[]={1.0,1.0,1.0};
static GLfloat c4[]={1.0,0.0,1.0};
static GLfloat c5[]={1.0,0.0,0.0};
static GLfloat c6[]={1.0,1.0,0.0};
static GLfloat c7[]={0.0,1.0,0.0};
static GLfloat c8[]={1.0,1.0,1.0};
```

- (4) 在 `PreCreateWindow` 函数中设置窗体样式, 代码如下:

```
BOOL COPENGLCubeView::PreCreateWindow(CREATESTRUCT& cs)
{
cs.style |= WS_CLIPSIBLINGS | WS_CLIPCHILDREN;
return CView::PreCreateWindow(cs);
}
```

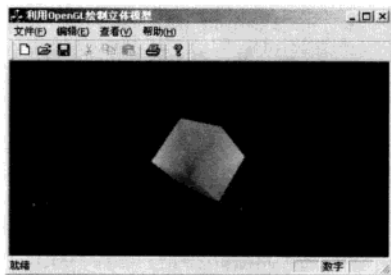


图 3.58 利用 OpenGL 绘制立体模型



(5) 在 OnDraw 函数中实现绘制, 代码如下:

```
void COPENGLCubeView::OnDraw(CDC* pDC)
{
 COPENGLCubeDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glPushMatrix();
 glRotatef(45,0.0,1.0,0.0);
 glRotatef(315,0.0,0.0,1.0);
 GLfloat light_ambient[] = {0.3,0.2,0.5};
 GLfloat light_diffuse[] = {1.0,1.0,1.0};
 GLfloat light_position[] = { 2.0, 2.0, 2.0, 1.0 };
 GLfloat light1_ambient[] = {0.3,0.3,0.2};
 GLfloat light1_diffuse[] = {1.0,1.0,1.0};
 GLfloat light1_position[] = { -2.0, -2.0, -2.0, 1.0 };
 glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
 glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
 glLightfv(GL_LIGHT0, GL_POSITION, light_position);
 glLightfv(GL_LIGHT1, GL_AMBIENT, light1_ambient);
 glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
 glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glEnable(GL_LIGHT1);
 glDepthFunc(GL_LESS);
 glEnable(GL_DEPTH_TEST);
 glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);
 glEnable(GL_COLOR_MATERIAL);
 glFlush();
 glBegin (GL_QUADS);
 glColor3fv(c1);
 glNormal3fv(m1);
 glVertex3fv(p1);
 glColor3fv(c2);
 glVertex3fv(p2);
 glColor3fv(c3);
 glVertex3fv(p3);
 glColor3fv(c4);
 glVertex3fv(p4);

 glColor3fv(c5);
 glNormal3fv(m5);
 glVertex3fv(p5);
 glColor3fv(c6);
 glVertex3fv(p6);
 glColor3fv(c7);
 glVertex3fv(p7);
 glColor3fv(c8);
 glVertex3fv(p8);

 glColor3fv(c5);
 glNormal3fv(m3);
 glVertex3fv(p5);
 glColor3fv(c6);
 glVertex3fv(p6);
 glColor3fv(c3);
 glVertex3fv(p3);
 glColor3fv(c4);
 glVertex3fv(p4);

 glColor3fv(c1);
 glNormal3fv(m4);
 glVertex3fv(p1);
 glColor3fv(c2);
 glVertex3fv(p2);
 glColor3fv(c7);
 glVertex3fv(p7);
 glColor3fv(c8);
 glVertex3fv(p8);

 glColor3fv(c2);
 glNormal3fv(m5);
 glVertex3fv(p2);
 glColor3fv(c3);
 glVertex3fv(p3);
 glColor3fv(c6);
 glVertex3fv(p6);
}
```

(6) 在 OnSize 函数中进行观看角度和视图大小等方面的设置, 代码如下:

```

{
 CView::OnSize(nType, cx, cy);
 if (0 >= cx || 0 >= cy)
 {
 return;
 }
 CView::OnSize(nType, cx, cy);
 ::glViewport(0, 0, cx, cy);
 ::glMatrixMode(GL_PROJECTION);
 ::glLoadIdentity();
 GLdouble dblaspect = (GLdouble)cx / (GLdouble)cy;
 ::glPerspective(40.0f, dblaspect, 1.f, 20.0f);
 ::glMatrixMode(GL_MODELVIEW);
 ::glLoadIdentity();
 ::glTranslatef(0.0f, 0.0f, -5.0f);
 ::glRotatef(20.0f, 1.0f, 0.0f, 0.0f);
}

```

```

{
 if (CView::OnCreate(lpCreateStruct) == -1)
 return -1;

 m_pContextDC=new CClientDC(this);
 ASSERT(m_pContextDC);
 static PIXELFORMATDESCRIPTOR pfd =
 {sizeof(PIXELFORMATDESCRIPTOR),1,
 PFD_DRAW_TO_WINDOW|PFD_SUPPORT_OPENGL|PFD_DOUBLEBUFFER,PFD_TYPE_RGBA,
 24, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 16, 0,
 0, PFD_MAIN_PLANE, 0, 0, 0, 0 };
 int pixelformat=::ChoosePixelFormat(m_pContextDC->GetSafeHdc(),&pfd);
 if(pixelformat==0)
 {
 AfxMessageBox("初始化失败");
 return -1;
 }
 SetPixelFormat(m_pContextDC->GetSafeHdc(),pixelformat,&pfd);
 m_hOglRC=wglCreateContext(m_pContextDC->GetSafeHdc());
 wglMakeCurrent(m_pContextDC->GetSafeHdc(),m_hOglRC);
 return 0;
}

```

```
{
 wglDeleteContext(m_hOpenglRC);
 delete m_pContextDC;
 return CView::DestroyWindow();
}
```

根据本实例，读者可以：

- 

## 实例 150

## 利用 OpenGL 绘制 NURBS 曲线

这是一个自娱自乐的实例

实例位置: 光盘\mingrisoft\03\150

### 实例说明

本实例通过对 OpenGL 库的引用, 实现了 NURBS 曲线的绘制, 如图 3.59 所示。

### 技术要点

本实例使用 `gluBeginSurface` 函数、`gluNurbsSurface` 函数和 `gluEndSurface` 函数来绘制 NURBS 曲线。这 3 个函数都需要使用 `gluNewNurbsRenderer` 函数生成对象, 该函数是绘制 NURBS 曲线的主要函数, 其语法如下:

```
void gluNurbsSurface(GLUnurbsObj *nobj, GLint sknot_count, GLfloat *sknot,
 GLint tknot_count, GLfloat *tknot, GLint s_stride, GLint t_stride,
 GLfloat *ctarray, GLint sorder, GLint torder, GLenum type);
```

参数说明:

- `nobj`: NURBS 曲线对象指针。
- `sknot_count`:  $U$  轴节点数量。
- `sknot`:  $U$  轴节点数组。
- `tknot_count`:  $V$  轴节点数量。
- `tknot`:  $V$  轴节点数组。
- `s_stride`: NURBS 对象数组在  $U$  轴的偏移数值。
- `t_stride`: NURBS 对象数组在  $V$  轴的偏移数值。
- `ctarray`: NURBS 对象数组。
- `sorder`:  $U$  轴表面参数。
- `torder`:  $V$  轴表面参数。
- `type`: 表面的类型。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在工程中加入对 `opengl32.lib`、`glu32.lib`、`glaux.lib` 库的引用。
- (3) 添加 OpenGL 头文件的引用, 代码如下:

```
#include "gl\gl.h"
#include "gl\glu.h"
#include "gl\glaux.h"
```

```
class COPENGLNURBSView : public CView
{
public:
 COPENGLNURBSDoc* GetDocument();
 CDC* m_pContextDC;
 //加入变量的声明
 HGLRC m_hOpenGLRC;
 CRect oldrc;
 GLUnurbsObj *nurb;
 GLfloat point[4][4][3];
}
```

- (4) 在 `PreCreateWindow` 函数中设置窗体样式, 代码如下:

```
BOOL COPENGLNURBSView::PreCreateWindow(CREATESTRUCT& cs)
{
```

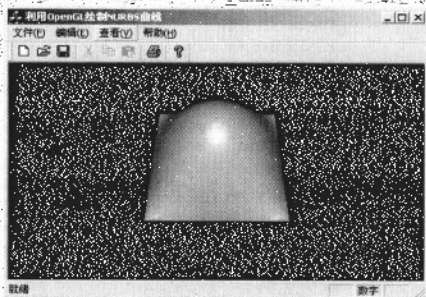


图 3.59 利用 OpenGL 绘制 NURBS 曲线

```
cs.style |= WS_CLIPSIBLINGS | WS_CLIPCHILDREN;
return CView::PreCreateWindow(cs);
}
```

(5) 在 OnDraw 函数中实现绘制 NURBS 曲线, 代码如下:

```
void COPENGLNURBSView::OnDraw(CDC* pDC)
{
 COPENGLNURBSDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);

 GLfloat diffuse[] = {0.88, 0.66, 0.22, 1.0};
 GLfloat specular[] = {0.92, 0.9, 0.0, 1.0};
 GLfloat shininess[] = {80.0};

 glClearColor(0.0, 0.0, 0.0, 1.0);
 glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);
 glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
 glMaterialfv(GL_FRONT, GL_SHININESS, shininess);

 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glDepthFunc(GL_LESS);
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_AUTO_NORMAL);
 glEnable(GL_NORMALIZE);

 int u, v;
 for (u=0; u<4; u++)
 {
 for (v=0; v<4; v++)
 {
 point[u][v][0] = 2.0*((GLfloat)u-1.5);
 point[u][v][1] = 2.0*((GLfloat)v-1.5);

 if (u==1||u==2)&&(v==1||v==2))
 point[u][v][2] = 3.0;
 else
 point[u][v][2] = -3.0;
 }
 }

 nurb = gluNewNurbsRenderer();
 gluNurbsProperty(nurb, GLU_SAMPLING_TOLERANCE, 25.0);
 gluNurbsProperty(nurb, GLU_DISPLAY_MODE, GLU_FILL);

 GLfloat knots[8] = {0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0};

 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

 glPushMatrix();
 glRotatef(330.0, 1.0, 0.0, 0.0);
 glScalef(0.5, 0.5, 0.5);

 gluBeginSurface(nurb);
 gluNurbsSurface(nurb, 8, knots, 8,
 knots, 4*3, 3, &point[0][0][0],
 4, 4, GL_MAP2_VERTEX_3);
 gluEndSurface(nurb);

 glPopMatrix();
 glFlush();

 if (FALSE == ::SwapBuffers(m_pContextDC->GetSafeHdc()))
 AfxMessageBox("交换缓冲区失败");
}
```

(6) 在 OnSize 函数中进行观看角度和视图大小等方面的设置, 代码如下:

```
void COPENGLNURBSView::OnSize(UINT nType, int cx, int cy)
{
 CView::OnSize(nType, cx, cy);
 if (0 >= cx || 0 >= cy)
 {
 return;
 }
 ::glViewport(0, 0, cx, cy);
 ::glMatrixMode(GL_PROJECTION);
 ::glLoadIdentity();
 GLdouble dblaspect = (GLdouble)cx / (GLdouble)cy;
```



```
::gluPerspective(45.0f,dblaspect,3.0f,8.0f);
::glMatrixMode(GL_MODELVIEW);
::glLoadIdentity();
::glTranslatef(0.0f,1.0f,-5.0f);
```

(7) 在 OnCreate 函数中实现像素的设置和设备环境的初始化，代码如下：

```
int COPENGLNURBSView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
 if (CView::OnCreate(lpCreateStruct) == -1)
 return -1;

 m_pContextDC=new CClientDC(this);
 ASSERT(m_pContextDC);
 static PIXELFORMATDESCRIPTOR pfd =
 {sizeof(PIXELFORMATDESCRIPTOR),1,
 PFD_DRAW_TO_WINDOW|PFD_SUPPORT_OPENGL|PFD_DOUBLEBUFFER,PFD_TYPE_RGBA,
 24,0,0,0,0,0,0,0,
 0,0,0,0,0,0,16,0,
 0,PFD_MAIN_PLANE,0,0,0,0};
 int pixelformat::ChoosePixelFormat(m_pContextDC->GetSafeHdc(),&pfd);
 if(pixelformat==0)
 {
 AfxMessageBox("初始化失败");
 return -1;
 }
 SetPixelFormat(m_pContextDC->GetSafeHdc(),pixelformat,&pfd);
 m_hOpenGLRC=wglCreateContext(m_pContextDC->GetSafeHdc());
 wglMakeCurrent(m_pContextDC->GetSafeHdc(),m_hOpenGLRC);
 return 0;
}
```

(8) 关闭窗口后关闭设备环境句柄，代码如下：

```
BOOL COPENGLNURBSView::DestroyWindow()
{
 wglDeleteContext(m_hOpenGLRC);
 delete m_pContextDC;
 return CView::DestroyWindow();
}
```

## 举一反三

根据本实例，读者可以：

- 利用 OpenGL 绘制 Bezier 网状曲面。

## 3.12 GDI+程序设计

OpenGL 是开发三维立体动画的主要函数库，通过该库很容易实现物体的三维立体效果，本节将通过几个实例介绍如何通过 OpenGL 开发应用程序。

### 实例 151 使用 GDI+显示 GIF 动画

这是一个自娱自乐的实例

实例位置：光盘\mingrisoft\03\151

#### 实例说明

在 Visual C++6.0 中，并没有现成的控件显示 GIF 图像，但是在 .NET 环境下，用户可以使用 GDI+方便地显示各种图像，包括 GIF 图像，本实例实现了 GDI+显示 GIF 图像，程序运行效果如图 3.60 所示。

#### 技术要点

有关如何在 Visual C++6.0 中使用 GDI+，请参考第 2 章实

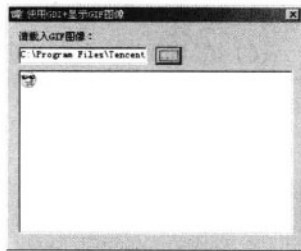


图 3.60 使用 GDI+显示 GIF 动画

例 016 显示 GIF 的 ATL 控件的技术要点部分。下面笔者介绍 GDI+库提供的位图对象——Bitmap。GDI+中的位图对象功能非常强大，它不仅可以显示位图文件、还可以显示 JPEG、GIF、PNG 等。通常，为了根据文件名获得一个位图对象，可以使用 Bitmap 的 FromFile 方法，该方法是一个静态方法，直接使用类名 Bitmap 即可调用。例如：

```
Bitmap* m_pBmp;
m_pBmp = Bitmap::FromFile(szFileName.AllocSysString());
```

为了获取图像的宽度和高度，在 Bitmap 对象中提供了 GetWidth 方法获取图像宽度，提供了 GetHeight 方法获取图像高度。在 Bitmap 对象中没有提供绘制图像的功能，在 GDI+中，绘制图像的功能被封装在 Graphics 中。在定义 Graphics 对象时，该对象可以关联一个设备上下文对象，然后调用 Graphics 对象的 DrawImage 方法就可以将某个图像输出到设备上下文中。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加编辑框、按钮、图片等控件。
- (3) 引用 Gdiplus.h 头文件，链接 GDI+库文件。代码如下：

```
#pragma comment(lib, "Gdiplus/GdiPlus.lib")
#include "Gdiplus/Gdiplus.h"
using namespace Gdiplus;
```

- (4) 向对话框类中添加成员变量。代码如下：

```
GdiplusStartupInput m_Gdiplus; //定义GDI+初始化变量
ULONG_PTR m_pGdiToken; //定义GDI+标识
Bitmap* m_pBmp; //定义位图对象
BOOL m_bLoaded; //是否已加载图像
int m_nFrameIndex; //记录当前显示帧索引
```

- (5) 在对话框初始化时开始一个计时器，初始化 GDI+库。代码如下：

```
SetTimer(1, 200, NULL); //开始一个计时器
GdiplusStartup(&m_pGdiToken, &m_Gdiplus, NULL); //初始化GDI+库
```

- (6) 处理 “...” 按钮的单击事件，从磁盘中加载 GIF 动画。代码如下：

```
void CShowGIFDlg::OnBrowne
{
 CFileDialog fDlg(TRUE, NULL, NULL, OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
 "Gif图片|*.gif|"); //定义文件打开对话框
 if (fDlg.DoModal() == IDOK)
 {
 CString szFileName = fDlg.GetPathName(); //获取文件名称
 m_GifName.SetWindowText(szFileName); //在编辑框中显示文件名称
 m_pBmp = Bitmap::FromFile(szFileName.AllocSysString()); //根据文件名称获取一个位图对象
 if (m_pBmp != NULL)
 {
 m_bLoaded = TRUE; //设置已加载标记
 m_nFrameIndex = 0; //初始化图像帧索引
 }
 }
}
```

- (7) 处理对话框的 WM\_TIMER 消息，如果已加载了 GIF 动画，则显示 GIF 动画。代码如下：

```
void CShowGIFDlg::OnTimer(UINT nIDEvent)
{
 if (m_bLoaded)
 {
 int nDimCount = m_pBmp->GetFrameDimensionsCount(); //获取帧维数
 GUID *pGuids = new GUID[nDimCount]; //获取标识Dimension的GUID
 m_pBmp->GetFrameDimensionsList(pGuids, nDimCount);
 int m_FrameCount = m_pBmp->GetFrameCount(pGuids); //获取第一个Dimension中的图像帧数
 UINT nSize = 0;
 UINT nDelay = PropertyTagFrameDelay;
 m_pBmp->GetPropertySize(nSize, &nDelay); //先计算属性相关数据的大小
 PropertyItem *pItem = NULL;
 pItem = (PropertyItem*)malloc(nSize);
 m_pBmp->GetAllPropertyItems(nSize, nDelay, pItem);
 nDelay = ((long*)pItem->value)[m_nFrameIndex]; //获取每一帧的时间间隔
 free(pItem);
 delete [] pGuids;
 CMemDC dc(m_DemoArea.GetDC(), CRect(0, 0,
 m_pBmp->GetWidth(), m_pBmp->GetHeight()));
```

```

Graphics graphic(dc.m_hDC); //显示图像帧
graphic.DrawImage(m_pBmp, 0, 0, m_pBmp->GetWidth(), m_pBmp->GetHeight());
if (m_nFrameIndex >= m_FrameCount - 1) //最后一帧显示结束, 重新显示图像帧
{
 m_nFrameIndex = 0;
}
GUID Guid = FrameDimensionTime;
m_pBmp->SelectActiveFrame(&Guid, m_nFrameIndex++); //显示下一帧
}
CDialog::OnTimer(nIDEvent);
}

```

### 举一反三

根据本实例, 读者可以:

- 利用 OpenGL 绘制位图变换。

## 实例 152

### 使用 GDI+实现图像格式转换

这是一个自娱自乐的实例

实例位置: 光盘\mingrisoft\03\152

### 实例说明

图片转换是指将指定文件格式的图片转换成另一种图片格式的文件。由于在不同的系统中所使用的图片格式各不相同, 为了使一张图片文件可以按不同的格式使用就必须将图片文件转换成其他类型的格式。本实例就是利用 GDI+实现了一个图片格式转换的工具, 如图 3.61 所示, 并且正方体每个顶点的颜色各不相同。

### 技术要点

本实例使用 GDI+所提供的功能来实现图片文件不同格式间的转换, GDI+是 GDI 图形库的一个增强版本, Visual C++可以使用这个库。它内建于 Windows XP 和 Microsoft .NET, 而对于 Windows 98、Windows NT 和 Windows 2000, 则有一个可重新发布的版本。GDI+是一个 Visual C++ API。它用 C++类和 Visual C++方法。为了使用 GDI+, 必须包含 (#include) <gdiplus.h>文件, 并将工程链接到 gdiplus.lib 库, 这两个文件包含在最新的 Windows SDK 中。

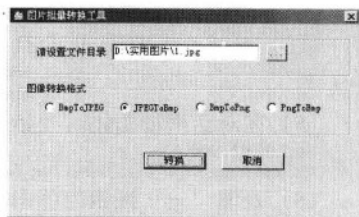


图 3.61 使用 GDI+实现图像格式转换

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个编辑框控件用来显示需要转换的图片文件路径。添加 4 个单选按钮控件用来设置图片文件转换的格式。添加两个按钮控件, 分别设置其 Caption 属性为“转换”和“取消”。
- (3) 定义名为 GetCodecClsid 的全局函数, 该函数用来获取指定文件类型的唯一标识。代码如下:

```

int GetCodecClsid(const WCHAR* format, CLSID* pClsid)
{
 UINT codenum = 0;
 UINT size = 0;
 ImageCodecInfo* pImageCodecInfo = NULL;
 GetImageEncodersSize(&codenum, &size);
 if (size == 0)
 return -1;
 pImageCodecInfo = new ImageCodecInfo[size];
 if (pImageCodecInfo == NULL)
 return -1;
 GetImageEncoders(codenum, size, pImageCodecInfo);
 for (UINT j = 0; j < codenum; ++j)
 {
 if (wcsncmp(pImageCodecInfo[j].MimeType, format) == 0)

```

```

 *pClsid = pImageCodecInfo[j].Clsid;
 delete []pImageCodecInfo;
 return 0;
 }
}
delete []pImageCodecInfo;
return -1;
}

```

(4) 在窗体类中实现 GetImageType 方法，该用法用来获取图片文件转换的格式类型。代码如下：

void CImageConvertDlg::GetImageType(enum IImageType &itSrc, enum IImageType &itDes)

```

{
 CButton* pBtn = NULL;
 for(UINT nID=IDC_BMPTOJPEG; nID<=IDC_PNGTOBMP; nID++)
 {
 pBtn = (CButton*) GetDlgItem(nID);
 if (pBtn != NULL)
 {
 //单选按钮被选中
 if (pBtn->GetCheck() > 0)
 {
 switch (nID)
 {
 case IDC_BMPTOJPEG:
 {
 itSrc = IT_BMP;
 itDes = IT_JPG;
 break;
 }
 case IDC_JPEGTOBMP:
 {
 itSrc = IT_JPG;
 itDes = IT_BMP;
 break;
 }
 case IDC_BMPTOPNG:
 {
 itSrc = IT_BMP;
 itDes = IT_PNG;
 break;
 }
 case IDC_PNGTOBMP:
 {
 itSrc = IT_PNG;
 itDes = IT_BMP;
 break;
 }
 }
 break;
 }
 }
 }
}

```

(5) 单击窗体上的“转换”按钮实现对图片的格式转换。代码如下：

void CImageConvertDlg::OnTransform()

```

{
 //获取文件目录
 CString szFileDir;
 m_FileDir.GetWindowText(szFileDir);
 if (szFileDir.IsEmpty())
 {
 MessageBox("请选择文件!", "提示");
 return;
 }
 //确定转换的源文件和目标文件的格式
 IImageType itSrc, itDes;
 GetImageType(itSrc, itDes);
 CLSID clsid;
 CString szSrcExt, szDesExt; //定义文件扩展名

 if (itSrc == IT_JPG)
 {
 szSrcExt = ".jpg";
 }
 else if (itSrc == IT_BMP)
 {
 szSrcExt = ".bmp";
 }
}

```



```

 }
 else if (itSrc == IT_PNG)
 {
 szSrcExt = "png";
 }

 if (itDes == IT_JPG)
 {
 GetCodecClsid(L"image/jpeg", &clsid);
 szDesExt = "jpg";
 }
 else if (itDes == IT_BMP)
 {
 GetCodecClsid(L"image/bmp", &clsid);
 szDesExt = "bmp";
 }
 else if (itDes == IT_PNG)
 {
 GetCodecClsid(L"image/png", &clsid);
 szDesExt = "png";
 }

 try
 {
 BOOL ret = TRUE;
 //定义压缩参数
 int nQuality = 95;
 EncoderParameters Encoders;
 Encoders.Count = 1;
 Encoders.Parameter[0].Guid = EncoderQuality;
 Encoders.Parameter[0].Type = EncoderParameterValueTypeLong;
 Encoders.Parameter[0].NumberOfValues = 1;
 Encoders.Parameter[0].Value = &nQuality;


 if (m_Strextend == szSrcExt)
 {
 //根据源位图文件构建一个GDI+位图对象
 Bitmap *pBmp = Bitmap::FromFile(szFileDir.AllocSysString());
 if (pBmp)
 {
 char chName[MAX_PATH] = {0};
 int pos = szFileDir.ReverseFind('\\');
 strcpy(chName, szFileDir.Left(pos));
 strcat(chName, "\\");
 strcat(chName, szDesExt);
 CreateDirectory(chName, NULL);
 strcat(chName, "\\");
 CString JpgFile = chName;
 JpgFile += m_FileName.Left(m_FileName.GetLength() - m_Strextend.GetLength());
 JpgFile += "."; //添加扩展名
 JpgFile += szDesExt;
 if (itDes == IT_JPG)
 {
 pBmp->Save(JpgFile.AllocSysString(), &clsid, &Encoders);
 }
 else
 {
 pBmp->Save(JpgFile.AllocSysString(), &clsid);
 }
 }
 delete pBmp; //释放位图对象
 MessageBox("转换成功!");
 }
 }
 catch(...)
 {
 MessageBox("转换失败!");
 }
}

```

## 举一反三

根据本实例，读者可以：

- 实现位图文件的批量转换。



## 第4章 多媒体技术

- 动画
- 制作与播放音频
- 多媒体控制
- 屏幕保护相关程序
- DirectShow 程序设计

Visual C++

## 4.1 动 画

在开发多媒体应用程序时,经常涉及动画的播放。本节将通过几个实例介绍常用动画文件的播放。

### 实例 153 屏幕动画精灵

本实例可以提高基础技能

实例位置: 光盘\mingrisoft\04\153

#### 实例说明

在 Office、瑞星等应用软件中,提供了一个动画精灵,即 Office 助手和瑞星小狮子,使程序增加了许多特色。在本例中,笔者也设计了一个类似的动画精灵,效果如图 4.1 所示。

#### 技术要点

许多读者都知道,使用微软公司的 Agent 控件可以显示一个动画精灵,该控件是一个 ActiveX 控件,用户可以在许多编程语言中使用。

Agent 控件的使用是非常简单的,但是如何设计.acs 文件呢。在微软的官方网站上提供了一个.acs 文件,其中定义了一些角色的动作。但是我们如何自己定义.acs 文件呢,例如实现向瑞星小狮子的效果。

微软提供了一个 Agent 助手编辑工具,即“Microsoft Agent Character Editor”,用户可以在微软的官方网站上找到,下面介绍使用 Agent 助手编辑工具设计.acs 文件的方法。

(1) 启动 Agent 助手编辑工具,在列表框中选择 character,在右边的属性页中选中 properties 页,在“Name”编辑框中输入角色名称,例如“MrAgent”。

(2) 在左边列表中选中“Animations”选项,在“File name”编辑框中设置动作的模板,即一个位图。然后在“Transparency color”选型中设置透明的颜色,本例为白色。因为我们的模板位图为白色背景。

(3) 鼠标右键单击 Animations 选项,在弹出的快捷菜单中选择“New Animation”菜单项新建一个动画,在 Animation Name 中输入 Move。

(4) 鼠标右键单击创建的动画——Move,在弹出的快捷菜单中选择“New Frame”菜单项,添加一帧。

(5) 按照步骤 4 的方式添加其他图像帧。

(6) 按照步骤 3 到步骤 5 的方式创建 Show 和 Hide 动画,并设置相应的图像帧。在创建 Show 动画时,在“Assign to State”列表选中“Showing”复选框,在创建 Hide 动画时,在“Assign to State”列表中选中“Hiding”复选框。

(7) 单击“File”菜单下的“Build Character”菜单项编译文件,将生成.acs 文件。

#### 实现过程

(1) 创建一个基于对话框的工程,工程名称为“Office”。



图 4.1 屏幕动画精灵

(2) 向对话框中添加按钮控件，并导入 Agent ActiveX 控件。

(3) 方法 OnInitDialog 是对话框初始化的实现，在对话框初始化时加载角色，并设置角色的右键弹出式菜单。其实现代码如下：

```
BOOL COfficeDlg::OnInitDialog()
{
 //...代码省略
 char szAppName[MAX_PATH] = {0};
 GetModuleFileName(NULL, szAppName, MAX_PATH); //获取文件名称
 char szDriver[128] = {0};
 char szDir[128] = {0};
 char szName[128] = {0};
 char szExt[128] = {0};
 _splitpath(szAppName, szDriver, szDir, szName, szExt); //分解目录
 char szFullPath[128] = {0};
 _makepath(szFullPath, szDriver, szDir, "Character1", "acs"); //组合目录
 COleVariant value1(szFullPath);
 m_Agent.GetCharacters().Load("MrAgent", value1); //加载角色
 m_Character = m_Agent.GetCharacters().Character("MrAgent"); //获取角色
 m_Character.SetAutoPopupMenu(FALSE); //隐藏默认的菜单
 IAgentCtlCommands pCommands;
 pCommands.AttachDispatch(m_Character.GetCommands());
 long enabled = 1;
 long visibled = 1;
 m_Agent.ShowOwnedPopups(FALSE); //隐藏弹出式菜单
 IAgentCtlCommandEx pCommand;
 pCommand.AttachDispatch(pCommands.Add("Move", COleVariant("表演(&A)"), COleVariant(""),
 COleVariant(enabled), COleVariant(visibled))); //添加菜单
 m_Menu.LoadMenu(IDR_MENU1); //加载菜单
 m_Agent.SetConnected(FALSE);
 return TRUE;
}
```

(4) 方法 OnShow 是“显示”按钮的单击事件实现，实现显示动画精灵。其实现代码如下：

```
void COfficeDlg::OnShow()
{
 m_Character = m_Agent.GetCharacters().Character("MrAgent");
 long prm = 0;
 COleVariant value(prm);
 m_Character.Show(value); //显示动画精灵
}
```

(5) 方法 OnAct 是“表演”按钮的单击事件实现，实现调用.acs 文件中的 Move 动作。其实现代码如下：

```
void COfficeDlg::OnAct()
{
 CString str = "Move";
 m_Character = m_Agent.GetCharacters().Character("MrAgent");
 m_Character.Play(str); //执行Move动作
}
```

(6) 方法 OnHide 是“隐藏”按钮的单击事件实现，实现隐藏桌面精灵。其实现代码如下：

```
void COfficeDlg::OnHide()
{
 m_Character = m_Agent.GetCharacters().Character("MrAgent");
 long prm = 0;
 COleVariant value(prm);
 m_Character.Hide(value); //隐藏动画精灵
}
```

## 举一反三

根据本实例，读者可以：

- 设计动画小程序。



## 实例 154 利用位图制作 AVI 动画

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\04\154

## 实例说明

AVI 文件是由一帧或多帧的图像构成的，按照不同的帧时间间隔播放，形成动画的效果。反之，用户也可以将一组图像数据按预定的时间间隔组合为一个 AVI 文件。本例就是使用 Visual C++ 来实现将 BMP 位图合成 AVI 文件。运行本实例，单击“...”按钮，选择保存 BMP 位图文件的文件夹，单击“生成 AVI”按钮，就可以将所选文件夹中的 BMP 位图合成为 AVI 文件了，如图 4.2 所示。

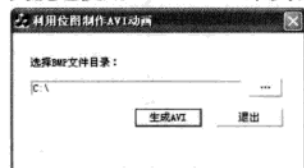


图 4.2 不透明的 Flash 动画

## 技术要点

本示例中实现将 BMP 位图合成 AVI 文件的功能时，主要用到了使用系统提供的一组 AVI 函数，下面对本实例中用到的关键技术进行详细讲解。

(1) AVIFileInit 函数。AVIFileInit 函数用于初始化 AVIFile 函数库。其语法格式如下：

```
STDAPL(VOID) AVIFileInit(VOID);
```

(2) AVIFileOpen 函数。AVIFileOpen 函数用于打开一个 AVI 文件，并返回文件的地址接口，其语法格式如下：

```
STDAPL AVIFileOpen(PAVIFILE * ppfile, LPCTSTR szFile, UINT mode, CLSID pclsidHandler);
```

- ppfile: 表示一个缓冲区指针，用于接收 IAVIFile 接口指针。
- szFile: 表示打开的文件名。
- mode: 表示打开文件时的访问模式，可选值如表 4.1 所示。

表 4.1 mode 参数值表

| 参 数 值               | 描 述                         |
|---------------------|-----------------------------|
| OF_CREATE           | 创建一个新的文件，如果文件已存在，删除文件中的所有数据 |
| OF_SHARE_DENY_NONE  | 不以排它的形式访问文件，其他进程可以读写文件      |
| OF_SHARE_DENY_READ  | 不以排它的形式访问文件，其他进程可以写文件       |
| OF_SHARE_DENY_WRITE | 不以排它的形式访问文件，其他进程可以读文件       |
| OF_SHARE_EXCLUSIVE  | 以排它的形式打开文件，其他进程不能访问文件       |
| OF_READ             | 以读的形式打开文件                   |
| OF_READWRITE        | 以读写的方式打开文件                  |
| OF_WRITE            | 以写的形式打开文件                   |

- pclsidHandler: 表示标准的或自定义的类标识符指针，如果为 NULL，系统将从注册表中选择一个默认类标识符。

(3) AVIFileCreateStream 函数。AVIFileCreateStream 函数用于在已存在的文件中创建一个流，并创建一个流接口，其语法格式如下：

```
STDAPL AVIFileCreateStream(PAVIFILE pfile, PAVISTREAM * ppavi, AVISTREAMINFO * psi);
```

- pfile: 表示打开的 AVI 文件句柄，通常从 AVIFileOpen 函数中获得。
- ppavi: 表示流接口指针。
- psi: 表示流信息的结构指针。

(4) AVIStreamSetFormat 函数。AVIStreamSetFormat 函数用于在指定的帧位置表示流格式，其语法格式如下：

```
STDAPL AVIStreamSetFormat(PAVISTREAM pavi, LONG lPos, LPVOID lpFormat, LONG cbFormat);
```

- pavi: 表示打开的流句柄。
- lPos: 表示流中的位置, 将在该位置处设置流格式。
- lpFormat: 表示新格式结构的指针。
- cbFormat: 表示 lpFormat 的大小。

(5) AVIStreamWrite 函数。AVIStreamWrite 函数用于向流中写入数据, 其语法格式如下:

```
STDAPL AVIStreamWrite(PAVISTREAM pavi, LONG lStart, LONG lSamples, LPVOID lpBuffer, LONG cbBuffer, DWORD dwFlags, LONG * plSampWritten, LONG * plBytesWritten);
```

- pavi: 表示打开的流句柄。
- lStart: 表示写入帧的起始位置。
- lSamples: 表示写入的帧数。
- lpBuffer: 表示存储写入数据的缓冲区。
- cbBuffer: 表示数据缓冲区的大小。
- dwFlags: 表示关联数据的标记, 如果为 AVIIF\_KEYFRAME, 表示关键帧。
- plSampWritten: 表示一个缓冲区指针, 用于接收写入的帧数, 可以为 NULL。
- plBytesWritten: 表示缓冲区 plSampWritten 的大小, 可以为 NULL。

(6) AVIFileRelease 函数。AVIFileRelease 函数用于结束 AVI 文件接口的引用计数, 如果引用计数为 0, 则关闭文件, 其语法格式如下:

```
STDAPL (ULONG) AVIFileRelease(PAVIFILE pfile);
```

- pfile: 表示打开的 AVI 文件句柄, 通常从 AVIFileOpen 函数中获得。

(7) AVIFileExit 函数。AVIFileExit 函数用于退出 AVIFile 函数库, 其语法格式如下:

```
STDAPL (VOID) AVIFileExit(VOID);
```

## 实现过程

- (1) 新建一个基于对话框的应用程序, 将其窗体标题改为“将 BMP 位图组合成 AVI 动画”。
- (2) 向对话框中添加 1 个静态文本控件、1 个编辑框控件和 3 个按钮控件。对话框中主要用到的控件及说明如表 4.2 所示。

表 4.2 对话框主要用到的控件及说明

| 控 件 ID      | 属 性 设 置               | 关 联 变 量        |
|-------------|-----------------------|----------------|
| IDC_STATIC  | Caption: 选择 BMP 文件目录; | 无              |
| IDC_EDIT1   | 无                     | CEdit m_BmpDir |
| IDC_BUTTON1 | Caption: ...          | 无              |
| IDOK        | Caption: 合成 AVI       | 无              |
| IDCANCEL    | Caption: 退出           | 无              |

(3) 处理“...”按钮的单击事件, 在该事件的处理函数调用“文件浏览”对话框选择 BMP 位图文件的存储位置, 其实现代码如下:

```
void CBuildAvidlg::OnButton1()
{
 char folder[MAX_PATH] = {0};
 BROWSEINFO binfo;
 memset(&binfo, 0, sizeof(BROWSEINFO));
 ITEMIDLIST* ilist;
 binfo.hwndOwner = 0;
 binfo.pidlRoot = NULL;
 binfo.pszDisplayName = folder;
 binfo.lpszTitle = "浏览文件";
 binfo.ulFlags = BIF_VALIDATE;
 binfo.lpfm = NULL;
 binfo.lParam = 0;
 ilist = SHBrowseForFolder(&binfo);
 SHGetPathFromIDList(ilist, folder);

 //定义一个字符串组
 //定义浏览信息
 //初始化binfo
 //定义一个ITEMIDLIST指针
 //设置对话框拥有者
 //设置根目录
 //设置显示名称
 //设置标题
 //设置标记
 //设置回调函数
 //设置回调函数的参数
 //显示浏览文件夹对话框
 //获取选择的文件夹名称
}
```

m\_BmpDir.SetWindowText(folder);

//设置编辑框文本

(4) 添加自定义函数 BmpsToAvi, 该函数用于将一组 BMP 位图合成为一个 AVI 文件, 其实现代码如下:

```
void CBuildAviDlg::BmpsToAvi(LPCSTR szFileName, LPCSTR szDir)
{
 CString BmpDir = szDir;
 BmpDir += _T("*.*");
 AVIFileInit();
 AVISTREAMINFO strhdr;
 PAVIFILE pFile;
 PAVISTREAM ps;
 PAVISTREAM pComStream;
 AVICOMPRESSOPTIONS pCompressOption;
 AVICOMPRESSOPTIONS FAR * opts[1] = {&pCompressOption};
 int nFrames = 0;
 CFileFind flFind;
 BOOL bret = flFind.FindFile(BmpDir);
 while(bret)
 {
 bret = flFind.FindNextFile();
 if(!flFind.IsDots() && !flFind.IsDirectory())
 {
 CString flname = flFind.GetFilePath();
 FILE *pf = fopen(flname, "rb");
 BITMAPFILEHEADER bmpFileHdr;
 BITMAPINFOHEADER bmpInfoHdr;
 fseek(pf, 0, SEEK_SET);
 fread(&bmpFileHdr, sizeof(BITMAPFILEHEADER), 1, pf);
 fread(&bmpInfoHdr, sizeof(BITMAPINFOHEADER), 1, pf);
 if(nFrames == 0)
 {
 //创建并打开avi文件
 AVIFileOpen(&pFile, szFileName, OF_WRITE | OF_CREATE, NULL);
 memset(&strhdr, 0, sizeof(strhdr));
 strhdr.fccType = streamtypeVIDEO;
 strhdr.fccHandler = 0;
 strhdr.dwScale = 1;
 strhdr.dwRate = 3;
 //设置图像代码
 strhdr.dwSuggestedBufferSize = bmpInfoHdr.biSizeImage;
 //设置显示区域
 SetRect(&strhdr.rcFrame, 0, 0, bmpInfoHdr.biWidth, bmpInfoHdr.biHeight);
 AVIFileCreateStream(pFile, &ps, &strhdr);
 opts[0]->fccType = streamtypeVIDEO;
 opts[0]->fccHandler = mmioStringToFOURCC("MSVC", 0);
 opts[0]->dwQuality = 7500;
 opts[0]->dwBytesPerSecond = 0;
 opts[0]->dwFlags = AVICOMPRESSF_VALID | AVICOMPRESSF_KEYFRAMES;
 opts[0]->lpFormat = 0;
 opts[0]->cbFormat = 0;
 opts[0]->dwInterleaveEvery = 0;
 AVIMakeCompressedStream(&pComStream, ps, &pCompressOption, NULL);
 AVIStreamSetFormat(pComStream, 0, &bmpInfoHdr, sizeof(BITMAPINFOHEADER));
 }
 BYTE *buffer = new BYTE[bmpInfoHdr.biWidth * bmpInfoHdr.biHeight * 3];
 fread(buffer, 1, bmpInfoHdr.biWidth * bmpInfoHdr.biHeight * 3, pf);
 AVIStreamWrite(pComStream, nFrames, 1, (LPBYTE)buffer, bmpInfoHdr.biSizeImage);
 nFrames++;
 fclose(pf);
 delete []buffer;
 }
 }
 AVIStreamClose(ps);
 AVIStreamClose(pComStream);
 if(pFile != NULL)
 AVIFileRelease(pFile);
 AVIFileExit();
}
```

(5) 处理“合成 AVI”按钮的单击事件, 在该事件的处理函数中通过“另存为”对话框设

置合成后的文件存储位置,并调用 BmpsToAvi 函数合成 AVI 文件,其实现代码如下:

```
void CBuildAviDlg::OnOK()
{
 CFileDialog fDlg(FALSE,"avi","demo",OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,
 "AVI文件|*.avi|",this); //定义文件保存对话框
 if(fDlg.DoModal()==IDOK) //判断用户是否按OK按钮
 {
 CString fname = fDlg.GetPathName(); //获取文件名称
 CString folder;
 m_BmpDir.GetWindowText(folder); //获得BMP位图存储位置
 BmpsToAvi(fname,folder); //合成AVI文件
 }
}
```

### 举一反三

根据本实例,读者可以:

- 利用图标制作动画;
- 电子相册屏幕保护程序。

## 实例 155 播放 GIF 动画


本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\04\155

### 实例说明

GIF 是流行于 Internet 上一种较为特殊的格式,即图像交换格式 (Graphics InterChange Format),此种文件格式具有以下几个特点。

- (1) 只支持 256 色以内的图像。
- (2) 采用无损压缩存储,在不影响图像质量的情况下,可以生成很小的文件。
- (3) 支持透明色,可以使图像浮现在背景之上。
- (4) 可以制作动画,这是最突出的一个特点。

本实例实现播放 GIF 动画,单击  按钮打开将要播放的 Gif 文件,如图 4.3 所示。

### 技术要点

本实例主要通过 WebBrowser 控件播放 Gif 动画,WebBrowser 控件需要通过选择菜单 Project/Add To Project/Components and Controls 添加到工程中。

通过 WebBrowser 控件的 Navigate2 方法可以实现浏览 GIF 动画。

### 实现过程

- (1) 新建一个名为 GifPlayer 的对话框 MFC 工程。
- (2) 在工程中添加浏览器控件库。
- (3) 在对话框上添加浏览器控件,添加成员变量 m\_gifplayer,添加按钮控件,设置 ID 属性为 DC\_BTADD, Caption 属性为“...”。

(4) 主要程序代码:

```
void CGifPlayerDlg::OnAdd()
{
 CFileDialog log(TRUE,"文件","*.gif",
 OFN_HIDEREADONLY,FILE(*.gif)*.gif|,NULL); //定义文件打开对话框
 if(log.DoModal()==IDOK)
```



图 4.3 播放 GIF 动画



```
{
 CString pathname=log.GetPathName();//获取文件路径
 m_gifplayer.Navigate2(COleVariant(pathname),NULL,NULL,NULL,NULL);
}
}
```

## 举一反三

根据本实例,读者可以:

- 制作一个循环播放的 GIF 文件。

## 实例 156 播放 Flash 动画

本实例可以提高基础技能

实例位置: 光盘\mingrisoft\04\156

## 实例说明

在互联网时代,读者一定不会对 Flash 感到陌生,这种格式的媒体文件是由 Macromedia 公司推出的交互式矢量图和 Web 动画的标准。使用此种格式的文件可以创作具有交互性的多媒体动画,并且此种文件非常的小。本实例就是实现 Flash 动画的播放,并使其背景颜色设为透明,运行程序,通过单击“打开”按钮打开一个 Flash 文件播放,非透明效果的 Flash 动画如图 4.4 所示。

## 技术要点

实例使用 Shockwave Flash Object 控件来播放,播放 Flash 需要使用 Shockwave Flash Object 控件,因为 Shockwave Flash Object 控件不是默认的控件,所以要向工程中添加该控件,添加 Shockwave Flash Object 控件的同时也会将 CShockwaveFlash 类添加到工程中。CShockwaveFlash 类的 LoadMovie 方法可以加载 Flash 文件,然后通过 Play 方法播放,通过 SetBackgroundColor 方法可以设置 Flash 文件透明显示。



图 4.4 播放 Flash 动画

## 实现过程

- (1) 新建基于对话框的应用程序。
- (2) 在对话框上添加 Button 控件,设置 ID 属性为 IDC\_BTADD,添加 Shockwave Flash Object 控件,添加成员变量 m\_flash。

- (3) 方法 OnAdd 是“打开”按钮的实现。

```
void CFlashPlayerDlg::OnAdd()
{
 CFileDialog dialog(true,"swf",NULL,OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,"Flash文件(*.swf)*.swf",this);
 if(dialog.DoModal()!=IDOK)
 {
 CString path=dialog.GetPathName();
 m_flash.LoadMovie(0,path);
 m_flash.SetBackgroundColor(::GetSysColor(COLOR_3DFACE));
 CRect rc;
 m_flash.GetClientRect(&rc);
 m_flash.Play();
 }
}
```

## 举一反三

根据本实例,读者可以:

- 设计动感界面、动画按钮。

## 实例 157 文字跟随鼠标

本实例可以提高基础技能

实例位置: 光盘\mingrisoft\04\157

## 实例说明

在使用浏览器浏览网页时,有时会遇到文字跟随的网页特效,鼠标移动到哪网上的浮动文字就跟随到哪。本实例就是在应用程序中模仿这个特效,效果如图 4.5 所示。

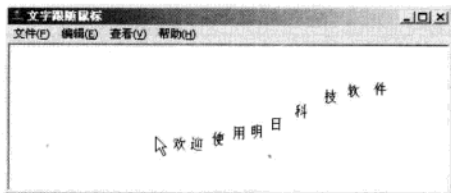


图 4.5 文字跟随鼠标

## 技术要点

在视图中输出文字使用 CDC 类的 TextOut 方法就可以实现,TextOut 方法可以设定文字的显示位置,在定时器中不断改变 TextOut 方法的参数就可以实现动态文字的显示,在定时器中根据鼠标的位置来设置文字的显示位置实现文字跟随鼠标移动的效果。

TextOut 方法可以在设备上下文中输出字符串。语法:

```
BOOL TextOut(int x, int y, const CString& str);
```

参数说明:

- x: 字符串输出起点的横坐标。
- y: 字符串输出起点的纵坐标。
- str: 将要显示的字符串。

## 实现过程

(1) 创建基于单文档视图结构应用程序。

(2) 在 OnDraw 方法中输出文字。

```
void CCharFollowView::OnDraw(CDC* pDC)
{
 CCharFollowDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 for(int i=0;i<10;i++)
 {
 pDC->TextOut(m_mousepoint[i].x,m_mousepoint[i].y,message[i]);
 }
}
```

(3) 在定时器 OnTimer 方法中变换文字的输出生位置。

```
void CCharFollowView::OnTimer(UINT nIDEvent)
{
 for(int i=9;i>=1;i--)
 {
 m_mousepoint[i].x=m_mousepoint[i-1].x+18;
 m_mousepoint[i].y=m_mousepoint[i-1].y;
 }
 m_mousepoint[0].x=m_point.x+18;
 m_mousepoint[0].y=m_point.y;
 Invalidate();
 CView::OnTimer(nIDEvent);
}
```

## 举一反三

根据本实例，读者可以：

- 设计电子时钟。

## 4.2 制作与播放音频

在制作多媒体软件时，不仅需要有丰富的界面，也需要动听的音乐。本节将介绍有关音频的录制和播放。

## 实例 158

## 可以选择播放曲目的 CD 播放器

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\04\158

## 实例说明

本实例实现一个可以列出 CD 中所有曲目的播放器。运行程序，单击“打开光驱”按钮可以将 CD 中所有的曲目显示在列表中，然后通过双击列表项来播放曲目，也可以通过单选列表项，然后单击“播放”按钮来播放。程序运行结果如图 4.6 所示。

## 技术要点

本实例通过使用 MCI 函数实现，其中 `mciSendCommand` 函数可以向多媒体设备发送命令，相应设备接收到命令后就会去实现相应的功能。播放 CD 需要许多这样的命令，其主要步骤是：首先向设备发送 `MCI_OPEN` 命令来打开设备，然后通过 `MCI_STATUS` 命令获取设备的状态，也就是检查光驱中是否有 CD 及 CD 中曲目的数量，最后通过 `MCI_PLAY` 命令实现 CD 曲目的播放。

## 实现过程

- (1) 新建一个名为 CDPlayer 的对话框 MFC 工程。
- (2) 在工程中加入对 `winmm.lib` 库的引用。
- (3) 在对话框上添加列表视图控件，设置 ID 属性为 `IDC_CDCATA`，添加成员变量 `m_cdcata`；添加 4 个按钮控件，设置 ID 属性分别为 `IDC_BTOPEN`、`IDC_PLAY`、`IDC_STOP`、和 `IDC_EXIT`，Caption 属性分别为“打开光驱”、“播放”、“停止”和“退出”。

- (4) CDPlayerDlg.cpp 文件中加入如下代码：

```
#define WM_ONTRAY WM_USER+1222 //自定义托盘消息
#include <MMSystem.h>
```

- (5) 在 `OnInitDialog` 函数中建立系统托盘，初始化列表控件，代码如下：

```
BOOL CCDPlayerDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...
 NOTIFYICONDATA data;定义托盘图标结构变量
 data.hWnd=m_hWnd; //指定消息接收窗口
 data.uCallbackMessage=WM_ONTRAY; //指定回调消息
 data.uFlags=NIF_MESSAGE|NIF_ICON; //处理标记
 data.hIcon=m_hIcon; //显示图标句柄
 data.uID=IDR_MAINFRAME;
 Shell_NotifyIcon(NIM_ADD,&data); //添加托盘图标
 m_cdcata.SetExtendedStyle(LVS_EX_CHECKBOXES|LVS_EX_GRIDLINES);
 m_cdcata.InsertColumn(0,"曲目",LVCFMT_LEFT,200);
 return TRUE;
}
```

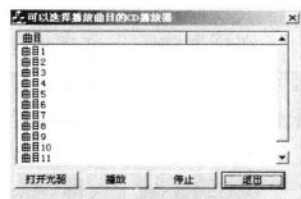


图 4.6 可以选择播放曲目的 CD 播放器

(6) “播放”按钮的实现函数，该函数用于播放选中的 CD 曲目，代码如下：

```
void CCDPlayerDlg::OnPlay()
{
 int i;
 i=m_cdcata.GetSelectionMark();//获取选中节点索引号
 if(i==1)return;
 MCI_SET_PARMS SetParms;
 SetParms.dwTimeFormat = MCI_FORMAT_TMSF;
 ::mciSendCommand(m_wDeviceID, MCI_SET,
 MCI_WAIT|MCI_SET_TIME_FORMAT,
 (DWORD)(LPVOID)&SetParms);
 ::mciSendCommand(m_wDeviceID, MCI_SEEK, MCI_SEEK_TO_START, NULL);
 MCI_PLAY_PARMS PlayParms;
 PlayParms.dwFrom=MCI_MAKE_TMSF(i,0,0,0);//播放
 ::mciSendCommand(m_wDeviceID, MCI_PLAY, MCI_FROM,
 (DWORD)(LPVOID)&PlayParms);
}
```

(7) “停止”按钮的实现函数，该函数用于停止播放 CD 曲目，代码如下：

```
void CCDPlayerDlg::OnStop()
{
 ::mciSendCommand(m_wDeviceID, MCI_STOP, NULL, NULL);//停止
}
```

(8) 添加 DestroyWindow 函数，在关闭窗体的时候关闭系统托盘和光驱设备，代码如下：

```
BOOL CCDPlayerDlg::Destroy Window()
{
 NOTIFYICONDATA data;
 data.cbSize=sizeof(NOTIFYICONDATA);
 data.hWnd=m_hWnd;
 data.uID=IDR_MAINFRAME;
 Shell_NotifyIcon(NIM_DELETE,&data);//删除托盘图标
 ::mciSendCommand(m_wDeviceID, MCI_CLOSE, MCI_WAIT, NULL);
 return CDialog::Destroy Window();
}
```

(9) “退出”按钮的实现函数，该函数用于退出应用程序，代码如下：

```
void CCDPlayerDlg::OnExit()
{
 this->OnCancel();//关闭窗口体
}
```

(10) 添加列表控件的 NM\_DBLCLK 消息的实现函数。实现双击列表项播放曲目，代码如下：

```
void CCDPlayerDlg::OnDbclickCdcata(NMHDR* pNMHDR, LRESULT* pResult)
{
 int i;
 i=m_cdcata.GetSelectionMark();//获取选中索引
 if(i==1)return;
 MCI_SET_PARMS SetParms;
 SetParms.dwTimeFormat = MCI_FORMAT_TMSF;
 ::mciSendCommand(m_wDeviceID, MCI_SET,
 MCI_WAIT|MCI_SET_TIME_FORMAT,
 (DWORD)(LPVOID)&SetParms);
 ::mciSendCommand(m_wDeviceID, MCI_SEEK, MCI_SEEK_TO_START, NULL);
 MCI_PLAY_PARMS PlayParms;
 PlayParms.dwFrom=MCI_MAKE_TMSF(i,0,0,0);
 ::mciSendCommand(m_wDeviceID, MCI_PLAY, MCI_FROM,
 (DWORD)(LPVOID)&PlayParms);//播放
 *pResult = 0;
}
```

(11) “打开光驱”按钮的实现函数，该函数主要实现打开 CD 设备，检测是否有 CD 光盘，并获得 CD 中的曲目数量，代码如下：

```
void CCDPlayerDlg::OnOpen()
{
 //打开系统中的CD设备
 MCI_OPEN_PARMS OpenParms;
 OpenParms.lpstrDeviceType = (LPCSTR) MCI_DEVTYPE_CD_AUDIO;
 int ireturn=::mciSendCommand(NULL,
 MCI_OPEN,
 MCI_WAIT|MCI_OPEN_SHAREABLE|
 MCI_OPEN_TYPE|MCI_OPEN_TYPE_ID,
 (DWORD)(LPVOID)&OpenParms);
 if(ireturn==0)
 {
 //获取CD设备的状态以查看是否有CD光盘
 m_wDeviceID=OpenParms.wDeviceID;
 MCI_STATUS_PARMS StatusParms;
```



```

StatusParms.dwItem=MCI_STATUS_MEDIA_PRESENT;
ireturn=:mciSendCommand(m_wDeviceID,
MCI_STATUS, MCI_STATUS_ITEM,
(DWORD)(LPVOID) &StatusParms);
if(ireturn==0)
{
 //获得CD中的曲目数量, 然后添加到列表中
 StatusParms.dwItem = MCI_STATUS_NUMBER_OF_TRACKS;
 mciSendCommand(m_wDeviceID,
 MCI_STATUS, MCI_STATUS_ITEM,
 (DWORD)(LPVOID) &StatusParms);
 UINT cdnum=StatusParms.dwReturn;
 for(int i=0;i<cdnum;i++)
 {
 CString cdstr;
 cdstr.Format("曲目%d",i+1);
 m_cdcat.InsertItem(i,cdstr);
 }
}
}
}

```

### 举一反三

根据本实例, 读者可以:

- 设计具有进度条的播放器。

## 实例 159

## 开发具有记忆功能的 MP3 播放器

本实例可以提高基础技能

实例位置: 光盘\mingrsoft\04\159

### 实例说明

许多多媒体播放软件在打开时能够加载上一次播放的歌曲列表, 它是如何实现的呢? 本实例实现具有记忆功能的 MP3 播放器, 效果如图 4.7 所示。

### 技术要点

使用 WritePrivateProfileString 函数将 MP3 列表保存到 ini 文件中。

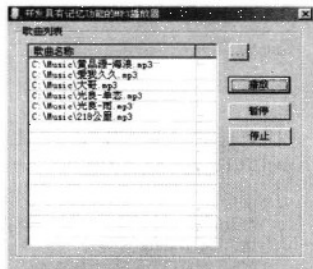


图 4.7 开发具有记忆功能的 MP3 播放器

WritePrivateProfileString 函数从 .ini 文件中获取指定节指定键名的字符串信息, 语法如下:

```

BOOL WritePrivateProfileString(LPCTSTR lpAppName,
LPCTSTR lpKeyName,LPCTSTR lpString,LPCTSTR lpFileName);

```

WritePrivateProfileString 函数中的参数说明如表 4.3 所示。

表 4.3 WritePrivateProfileString 函数中的参数说明

| 设置值              | 描述                                              |
|------------------|-------------------------------------------------|
| lpAppName        | 将要获取字符串数据所在的节名                                  |
| lpKeyName        | 将要获取字符串数据所在的键名                                  |
| lpDefault        | 如果没有找到键名, 函数将返回此值                               |
| lpReturnedString | 存放返回值的字符串指针                                     |
| nSize            | 设置将要保存的字符串的大小                                   |
| lpFileName       | ini 文件名, 可以是全路径, 如果不是全路径, 默认就在系统文件夹下新建一个 ini 文件 |

### 实现过程

- (1) 创建基于对话框的应用程序。

(2) 在对话框中添加列表控件和按钮控件。

(3) 方法 OnCancel 是退出应用程序时调用的函数, 在该函数内实现播放列表的保存。

void CMP3PlayerDlg::OnCancel()

```
{
 int num = m_Songs.GetItemCount();
 CString songname;
 CString key = "数量";
 CString songnum;
 songnum.Format("%d", num);

 CString str;
 GetModuleFileName(NULL, str.GetBuffer(0), MAX_PATH);
 int pos = str.ReverseFind("\\");
 CString temp = str;
 CString filename = temp.Left(pos);

 WritePrivateProfileString("歌曲列表", key.GetBuffer(0), songnum.GetBuffer(0), filename+"\\song.ini");
 for (int i = 0; i < num; i++)
 {
 key.Format("%d", i);
 songname = m_Songs.GetItemText(i, 0);
 WritePrivateProfileString("歌曲列表", key.GetBuffer(0), songname.GetBuffer(0), filename+"\\song.ini");
 }

 CDialog::OnCancel();
}
```

### 举一反三

根据本实例, 读者可以:

设计具有记忆功能的列表框。

## 实例 160 声音录制与播放

本实例是一个人性化的实例

实例位置: 光盘\mingrsoft\04\160

### 实例说明

本实例实现声音的录制和播放。运行程序, 如图 4.8 所示, 单击“录音”按钮后可以通过麦克风进行录音, 录音完成后须单击“停止”按钮, 如果要听录音的结果可以单击“播放”按钮。

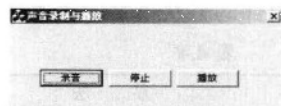


图 4.8 声音录制与播放

### 技术要点

本实例主要使用 MCI 函数进行声音的录制和播放。通过 MCIWndCreate 函数创建一个窗体句柄, 主要在该窗体中实现声音的录制; 创建窗体句柄后通过 MCIWndNew 函数打开录音设备, 通过 MCIWndCanRecord 函数判断是否可以录音, 如果可以录音, 则通过 MCIWndRecord 函数进行声音的录制。播放录音可以使用 MCIWndPlay 函数。

### 实现过程

(1) 新建一个名为 RecordSound 的对话框 MFC 工程。

(2) 在对话框上添加 3 个按钮控件, 设置 ID 属性分别为 IDC\_BTRECORD、IDC\_BTSTOP 和 IDC\_BTPLAY, Caption 属性分别为“录音”、“停止”和“播放”。

(3) 在 StdAfx.h 中加入下面语句:

```
#include <vfw.h>
#pragma comment(lib, "vfw32.lib")
```

(4) 在 RecordSoundDlg.h 文件中加入变量声明:

HWND mciwav;

(5) “录音”按钮的实现函数, 该函数用于开始录制声音, 代码如下:

```
void CRecordSoundDlg::OnRecord()
{
 MCIWndClose(mciwav);
 mciwav=MCIWndCreate(this->m_hWnd,::AfxGetApp()->m_hInstance,
 WS_CAPTION,NULL);
 MCIWndNew(mciwav,"waveaudio");
 if(MCIWndCanRecord(mciwav))
 MCIWndRecord(mciwav);
}
```

(6) “停止”的实现函数, 该函数用于停止录制声音, 代码如下:

```
void CRecordSoundDlg::OnPlay()
{
 if(MCIWndCanPlay(mciwav))
 MCIWndPlay(mciwav);
}
```

(7) “播放”按钮的实现函数, 该函数用于播放已录制的声音, 代码如下:

```
void CRecordSoundDlg::OnStop()
{
 MCIWndStop(mciwav);
}
```

### 举一反三

根据本实例, 读者可以:

- 实现声音的传送。

## 实例 161 制作 RealOne 播放器

本实例是一个人性化的实例

实例位置: 光盘\mingrisoft\04\161

### 实例说明

RealOne 是播放流媒体文件的主要工具, 已被广泛应用。本实例将实现能够播放 rm 文件的简单的 RealOne 播放器, 如图 4.9 所示。

### 技术要点

实现本实例需要添加 RealPlayer 组件, 系统中安装 Realplayer 播放器后, RealPlayer 组件才能添加到 VC 的工程中, RealPlayer 组件的方法很多, 主要方法介绍如表 4.4 所示。



图 4.9 制作 RealOne 播放器

表 4.4 RealPlayer 组件的主要方法

| 方 法                | 描 述    |
|--------------------|--------|
| DoPlay             | 播放     |
| DoPause            | 暂停     |
| DoStop             | 停止     |
| SetVolume          | 设置声音   |
| SetFullScreen      | 设置全屏   |
| SetBackgroundColor | 设置背景颜色 |
| GetPosition        | 获得位置   |
| GetPlayState       | 获得播放状态 |

### 实现过程

- (1) 新建一个名为 realplayerDlg 的对话框 MFC 工程。
- (2) 在工程中添加 realplayer 控件库。
- (3) 在对话框上添加图片控件, 设置 ID 属性为 IDC\_REALPLAYER; 添加 7 个按钮控件,



设置 ID 属性分别为 IDC\_BTOPEN、IDC\_BTPAUSE、IDC\_BTSTOP、IDC\_BT SOUNDADD、IDC\_BT SOUND SUB、IDC\_BT PROCADD 和 IDC\_BT PROC SUB, Caption 属性分别为“打开”、“暂停”、“停止”、“音量+”、“音量-”、“快进>>”和“快退<<”。

(4) “打开”按钮的实现函数, 该函数用于打开一个 RM 文件并播放该文件, 代码如下:

```
void CRealplayerDlgDlg::OnOpen()
{
 CString strname;
 CFileDialog dlg(TRUE, NULL, NULL, OFN_HIDEREADONLY, "realplay文件*.rm||");
 if(dlg.DoModal()==IDOK){
 strname=dlg.GetPathName();
 }
 if(strname!="")
 {
 m_realplayer.SetSource(strname);//设置播放文件
 m_realplayer.DoPlay();//播放文件
 }
}
```

(5) “暂停”按钮的实现函数, 该函数用于暂停播放, 代码如下:

```
void CRealplayerDlgDlg::OnPause()
{
 if(m_realplayer.GetPlayState()==0)return;
 CString text;
 GetDlgItem(IDC_BUTTON2)->GetWindowText(text);
 if(text=="暂停")
 {
 m_realplayer.DoPause();
 GetDlgItem(IDC_BUTTON2)->SetWindowText("播放");
 }
 else
 {
 m_realplayer.DoPlay();
 GetDlgItem(IDC_BUTTON2)->SetWindowText("暂停");
 }
}
```

(6) “停止”按钮的实现函数, 该函数用于停止播放, 代码如下:

```
void CRealplayerDlgDlg::OnStop()
{
 m_realplayer.DoStop();
}
```

(7) “音量+”按钮的实现函数, 该函数用于增加播放音量, 代码如下:

```
void CRealplayerDlgDlg::OnSoundAdd()
{
 short vl;
 if(m_realplayer.GetPlayState()==3)
 {
 vl=m_realplayer.GetVolume();
 if(vl<=90)
 {
 m_realplayer.DoPause();//暂停
 m_realplayer.SetVolume(vl+10);//增加音量
 m_realplayer.DoPlay();//播放
 }
 }
 else
 {
 if(vl<=90)
 m_realplayer.SetVolume(vl+10);
 }
}
```

(8) “音量-”按钮的实现函数, 该函数用于减小播放音量, 代码如下:

```
void CRealplayerDlgDlg::OnSoundSub()
{
 short vl;
 if(m_realplayer.GetPlayState()==3)
 {
 vl=m_realplayer.GetVolume();
 if(vl>=10)
 {
 m_realplayer.DoPause();//暂停
 m_realplayer.SetVolume(vl-10);//减小音量
 }
 }
}
```



```

 m_realplayer.DoPlay();//播放
 }
}
else
{
 if(v1>=10)
 m_realplayer.SetVolume(v1-10);
}
}

```

(9) “快进>>”按钮的实现函数，该函数用于播放当前时间增加两秒后的内容，代码如下：

```

void CRealplayerDlgDlg::OnProcAdd()
{
 if(m_realplayer.GetPlayState()==3)
 {
 long pos=m_realplayer.GetPosition();
 m_realplayer.DoPause();
 m_realplayer.SetPosition(pos+2000);//修改当前播放点
 m_realplayer.DoPlay();
 }
}

```

(10) “快退<<”按钮的实现函数，该函数用于播放当前时间减少两秒后的内容，代码如下：

```

void CRealplayerDlgDlg::OnProcSub()
{
 if(m_realplayer.GetPlayState()==3)
 {
 long pos=m_realplayer.GetPosition();
 if(pos<2000)return;
 m_realplayer.DoPause();
 m_realplayer.SetPosition(pos-2000);//修改当前播放点
 m_realplayer.DoPlay();
 }
}

```

### 举一反三

根据本实例，读者可以：

- 开发网络 RealOne 播放器。

## 4.3 多媒体控制

在设计多媒体应用程序时，经常需要对媒体设备进行各种控制。本节通过几个实例介绍如何进行多媒体控制。

### 实例 162 音频波形显示

本实例可以提高基础技能

实例位置：光盘\mingrisoft\04\162

### 实例说明

网上的许多媒体播放器都具有显示音频波形的特效，增强了播放器的视觉效果。在本例中，笔者通过捕获音频数据实现了音频波形显示，界面效果如图 4.10 所示。

### 技术要点

实现音频波形主要通过两个步骤完成，第一个步骤是捕获音频数据，第二个步骤是对音频数据进行傅立叶变换，将变换后的数据以图像的形式显示。

#### (1) 捕捉音频数据

实例捕捉声卡的波形。首先调用 `waveInOpen` 函数打开录音设备，然后调用 `waveInPrepare Header` 函数为录音设备准备缓冲区，最后调用 `waveInAdd`



图 4.10 可以选择播放曲目的 CD 播放器

Buffer 函数实现录音。

## (2) 傅立叶变换

傅立叶变换是数字信号处理中最基础的运算,被广泛应用于通信、医学、天文学等领域。在进行图像处理时,经常使用傅立叶变换。

## 实现过程

- (1) 创建一个基于对话框的应用程序。
- (2) 向对话框中添加按钮和图片控件。
- (3) 在 MP3PlayerDlg.h 头文件中添加 MMSystem.h 头文件和 winmm.lib 库文件的引用。
- (4) 在 CMP3PlayerDlg 类中添加 HWAVEIN、HWAVEOUT、WAVEFORMATEX 和 WAVEHDR 等类型的成员。

(5) 方法 RecordAudio 是“开始”按钮的实现,单击“开始”按钮后开始显示音频数据的波形。

```
void CMP3PlayerDlg::RecordAudio()
{
 static BOOL bChange = TRUE;
 memset(lpInbuf, 0, 1024*4);
 if (bChange == TRUE)
 {
 waveInUnprepareHeader(m_hWaveIn, &lpInWaveHdr[0], sizeof(WAVEHDR));

 waveInPrepareHeader(m_hWaveIn, &lpInWaveHdr[0], sizeof(WAVEHDR));
 MMRESULT mmRet = waveInAddBuffer(m_hWaveIn, &lpInWaveHdr[0], sizeof(WAVEHDR));
 if (mmRet != MMSYSERR_NOERROR)
 {
 return;
 }
 short int* pData = (short int*)lpInWaveHdr[0].lpData;
 for(int i=0; i<1024; i++)
 {
 m_SrcData[i].fReal = (DWORD)(*pData - 32768);
 m_SrcData[i].fImage = 0;
 pData ++;
 }
 bChange = FALSE;
 }
 else
 {
 waveInUnprepareHeader(m_hWaveIn, &lpInWaveHdr[1], sizeof(WAVEHDR));
 waveInPrepareHeader(m_hWaveIn, &lpInWaveHdr[1], sizeof(WAVEHDR));
 MMRESULT mmRet = waveInAddBuffer(m_hWaveIn, &lpInWaveHdr[1], sizeof(WAVEHDR));
 if (mmRet != MMSYSERR_NOERROR)
 {
 return;
 }
 short int* pData = (short int*)lpInWaveHdr[1].lpData;
 for(int i=0; i<1024; i++)
 {
 m_SrcData[i].fReal = (DWORD)(*pData - 32768);
 m_SrcData[i].fImage = 0;
 pData ++;
 }
 }
}

FFT(m_SrcData, m_DesData, 1024, 2*3.1415926);

CDC* pDC = m_AudioGraph.GetDC();
CRect clientRC;
m_AudioGraph.GetClientRect(clientRC);
int nClientHeight = clientRC.Height();
int nClientWidth = clientRC.Width();
int x = 0;
CBrush brush(RGB(255, 255, 255));
pDC->FillRect(clientRC, &brush);
brush.DeleteObject();
CPen pen(PS_SOLID, 1, RGB(0, 66, 33));
pDC->SelectObject(&pen);
```

```
for(int i=1; i<256; i++)
{
 if (x < nClientWidth)
 {
 int nHeight = sqrt((m_DesData[i].fReal * m_DesData[i].fReal
 + m_DesData[i].fImage * m_DesData[i].fImage)/1024);
 nHeight = nHeight < 8192? nHeight: 8192;
 nHeight = nClientHeight - nHeight * nClientHeight/8192;

 CRect rc(x, nHeight, x+1, nClientHeight-1);

 pDC->Rectangle(rc);
 x += 2;
 }
 else
 {
 break;
 }
}
pen.DeleteObject();
m_AudioGraph.ReleaseDC(pDC);
}
```

### 举一反三

根据本实例，读者可以：

- 在播放器中增加波形显示功能。

## 实例 163 利用 PC 喇叭播放声音

这是一个自娱自乐的实例

实例位置：光盘\mingrisoft\04\163

### 实例说明

PC 喇叭是 PC 上的一种发声设备，该设备由计算机中的硬件控制，需要发声时使用脉冲控制声音的声调和延时。软件中经常需要提示音来提醒用户的操作，目前通常是用声卡来完成的，而一些 PC 机上没有安装声卡，这样就用到 PC 喇叭发出声音来提醒用户。运行程序，如图 4.11 所示。

### 技术要点

使 PC 喇叭播放声音主要使用了 Beep 函数，语法如下：

```
BOOL Beep(DWORD dwFreq,DWORD dwDuration);
```

参数说明：

- dwFreq：指定频率，单位是赫兹，范围是从 37~32 767。
- dwDuration：持续实现，单位是毫秒。

使用 Beep 函数按照一定的音高和音长控制 PC 喇叭发出声音，还可以使 PC 喇叭播放音乐。当然这种情况下播放出的音乐是连续的单音，既无声部也无和弦。

每个音符的音高由声音的频率值确定，音长由这个音符的持续时间确定。例如中音 5 的频率为 392Hz，中音 5 发声 0.5s，函数调用为 Beep(392,500)，如表 4.5 所示。

表 4.5 音符和频率的对应关系

| 音 符 | 频 率 | 音 符 | 频 率 | 音 符 | 频 率   | 音 符 | 频 率    |
|-----|-----|-----|-----|-----|-------|-----|--------|
| 1   | 131 | 1   | 262 | 1   | 523.3 | 1   | 1046.5 |
| 2   | 147 | 2   | 296 | 2   | 587.3 | 2   | 1174.7 |

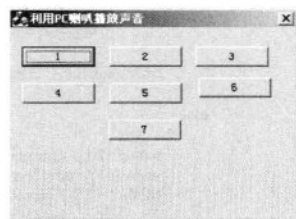


图 4.11 利用 PC 喇叭播放声音

续表

| 音 符 | 频 率 | 音 符 | 频 率   | 音 符 | 频 率   | 音 符 | 频 率    |
|-----|-----|-----|-------|-----|-------|-----|--------|
| 3   | 165 | 3   | 329.7 | 3   | 659.3 | 3   | 1318.5 |
| 4   | 176 | 4   | 349.2 | 4   | 698.5 | 4   | 1396.9 |
| 5   | 196 | 5   | 392   | 5   | 784   | 5   | 1568   |
| 6   | 220 | 6   | 440   | 6   | 880   | 6   | 1760   |
| 7   | 247 | 7   | 493.9 | 7   | 987.8 | 7   | 1975.5 |

## 实现过程

(1) 新建一个名为 PCSound 的对话框 MFC 工程。

(2) 在对话框上添加 7 个按钮控件。

(3) 主要程序代码：

```
void CPCSoundDlg::OnOne()
{
 ::Beep(264,500);
}

void CPCSoundDlg::OnTwo()
{
 ::Beep(296,500);
}

void CPCSoundDlg::OnThree()
{
 ::Beep(330,500);
}

void CPCSoundDlg::OnFour()
{
 ::Beep(349,500);
}

void CPCSoundDlg::OnFive()
{
 ::Beep(392,500);
}

void CPCSoundDlg::OnSix()
{
 ::Beep(440,500);
}

void CPCSoundDlg::OnSeven()
{
 ::Beep(494,500);
}
```

(4) 添加对话框 PertranslateMessage 消息实现函数，代码如下：

```
BOOL CPCSoundDlg::PreTranslateMessage(MSG* pMsg)
{
 if(pMsg->message==WM_KEYDOWN&&pMsg->wParam>VK_NUMPAD1)
 this->OnOne();
 if(pMsg->message==WM_KEYDOWN&&pMsg->wParam>VK_NUMPAD2)
 this->OnTwo();
 if(pMsg->message==WM_KEYDOWN&&pMsg->wParam>VK_NUMPAD3)
 this->OnThree();
 if(pMsg->message==WM_KEYDOWN&&pMsg->wParam>VK_NUMPAD4)
 this->OnFour();
 if(pMsg->message==WM_KEYDOWN&&pMsg->wParam>VK_NUMPAD5)
 this->OnFive();
 if(pMsg->message==WM_KEYDOWN&&pMsg->wParam>VK_NUMPAD6)
 this->OnSix();
 if(pMsg->message==WM_KEYDOWN&&pMsg->wParam>VK_NUMPAD7)
 this->OnSeven();
}
```



```
return CDialog::PreTranslateMessage(pMsg);
```

## 举一反三

根据本实例，读者可以：

- 将 1~7 键设置成从低音到高音的音符；
- 使用 PC 喇叭播放音乐。

## 实例 164 控制左右声道

本实例可以提高工作效率

实例位置：光盘\mingrisoft\04\164

## 实例说明

本实例实现控制系统的左右声道。运行程序，在程序中有两个滑动条分别是用来设置左声道和右声道的，如图 4.12 所示，由左向右拖动滑动条可以控制系统左声道或右声道的音量由小向大变化。

## 技术要点

本实例主要通过 `waveOutGetVolume` 函数来获得系统的音量，通过 `waveOutSetVolume` 函数来设置系统的音量。系统音量是一个 `DWORD` 值，它的前两个字节表示左声道的音量，后两个字节表示右声道的音量。系统当前音量和 `0x0000FFFF` 值做与运算可以得到系统左声道的音量；系统当前音量和 `0xFFFF0000` 值做与运算可以得到系统右声道的音量。

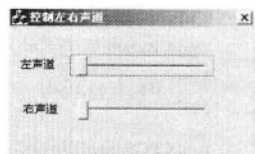


图 4.12 控制左右声道

## 实现过程

- (1) 新建一个名为 `LRSoundControl` 的对话框 MFC 工程。
- (2) 在对话框上添加两个滑块控件，设置 `ID` 属性分别为 `IDC_LEFT` 和 `IDC_RIGHT`，添加成员变量 `m_left` 和 `m_right`。

- (3) `StdAfx.h` 文件中加大如下代码：

```
#include <mmsystem.h>
#pragma comment(lib, "winmm.lib")
```

- (4) 在 `OnInitDialog` 函数中设置 Slider 控件的范围，代码如下：

```
BOOL CLRSoundControlDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 SetIcon(m_hIcon, TRUE);
 SetIcon(m_hIcon, FALSE);

 m_left.SetRange(0,200); //设置滑块的范围
 m_right.SetRange(0,200);
 return TRUE;
}
```

- (5) 添加 `WM_HSCROLL` 消息的处理函数，实现左声道右声道的控制，代码如下：

```
void CLRSoundControlDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
 DWORD pos;
 int scrollpos;
 if(pScrollBar->m_hWnd==m_left.m_hWnd)
 {
 scrollpos=m_left.GetPos(); //获取滑块位置
 ::waveOutGetVolume(0,&pos); //获取音量大小
 pos=pos&0x0000ffff|(scrollpos<<8);
 ::waveOutSetVolume(0,pos); //设置音量大小
 }
}
```

```

if(pScrollBar->m_hWnd==m_right.m_hWnd)
{
 scrollpos=m_right.GetPos();
 ::waveOutGetVolume(0,&pos);
 pos=pos&0xffff0000|(scrollpos<<24);
 ::waveOutSetVolume(0,pos);
}
CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}

```

### 举一反三

根据本实例，读者可以：

- 关闭和打开声道。

## 4.4 屏幕保护相关程序

屏幕保护程序可以用变换的颜色、图形或图像来防止屏幕荧光粉被损伤，当鼠标移动或按下任意键时能够终止它。下面通过几个实例介绍屏幕保护程序的设计。

### 实例 165 电子相册屏幕保护程序

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\04\165

### 实例说明

本实例实现一个具有个性化的电子相册屏幕保护程序。屏幕保护是由系统启动的、特殊的可执行文件，所以要将本实例应用程序的扩展名 exe 改为 src，并将程序放到系统的 system32 文件夹下，然后通过设置显示属性选中这个屏幕保护程序。屏幕保护运行结果如图 4.13 所示。



图 4.13 电子相册屏幕保护程序

### 技术要点

制作屏幕保护程序首先要将程序的对话框设置为没有标题栏，然后在启动后将对话框设置为全屏显示，并将背景颜色设置为黑色，接着就是在对话框上绘制图片了，最后还要对鼠标的移动、鼠标和键盘的按键消息进行处理，如果有这些消息发生就退出屏幕保护。将程序全屏显示需要使用 GetSystemMetrics 函数获得屏幕的宽度和高度，然后通过 MoveWindow 方法将程序全屏显示。

### 实现过程

- (1) 新建一个名为 Sreensavealbum 的对话框 MFC 工程。
- (2) 将对话框的 Border 属性改为 None。
- (3) 在头文件添加变量声明和自定义函数声明，代码如下：

```

CPoint curpt;
int x,y;
void DrawBitmap(CDC &dc, int nIndexBit);

```

- (4) 在 OnInitDialog 函数中实现变量的初始化和窗体大小的初始化，代码如下：

```

BOOL CSreensavealbumDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...
 ShowCursor(false);
 CRect rc(0,0,GetSystemMetrics(SM_CXSCREEN),GetSystemMetrics(SM_CYSCREEN));
 ::GetCursorPos(&curpt);
}

```

```

this->MoveWindow(rc); //移动窗口
SetTimer(1,500,NULL);
x=0;y=0;
return TRUE;
}

```

(5) 设置定时器, 在定时器内通过自定义函数绘制图片, 代码如下:

```
void CScreensavealbumDlg::OnTimer(UINT nIDEvent)
```

```

{
 KillTimer(1);
 CClientDC dc(this);
 static nIndexBit=0;
 if(nIndexBit>3)
 nIndexBit=0;
 DrawBitmap(dc,nIndexBit++); //绘制图像
 SetTimer(1,500,NULL);
 CDialog::OnTimer(nIDEvent);
}

```

(6) 自定义函数 DrawBitmap, 实现图片的绘制, 代码如下:

```
void CScreensavealbumDlg::DrawBitmap(CDC &dc, int nIndexBit)
```

```

{
 CDC dcmem;
 dcmem.CreateCompatibleDC(&dc);
 CBitmap m_Bitmap;
 m_Bitmap.LoadBitmap(IDB_BITMAP1+nIndexBit); //载入位图
 dcmem.SelectObject(m_Bitmap); //选中位图
 BITMAP bmp;
 GetObject(m_Bitmap,sizeof(bmp),&bmp); //获取位图对象
 int iscreenx=GetSystemMetrics(SM_CXSCREEN); //获取屏幕宽度
 int iscreeny=GetSystemMetrics(SM_CYSCREEN); //获取屏幕高度
 if(x>iscreenx)x=0;
 if(y>iscreeny)y=0;
 dc.BitBlt(x,y,bmp.bmWidth,bmp.bmHeight,&dcmem,0,0,SRCCOPY);
 Sleep(2000);
 dc.BitBlt(x,y,iscreenx,iscreeny,&dcmem,0,0,BLACKNESS);
 x+=80;
 y+=20;
 dcmem.DeleteDC();
}

```

(7) 添加处理鼠标和键盘按键消息的实现函数, 有相应的消息产生则关闭程序, 代码如下:

```
void CScreensavealbumDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
```

```

{
 PostMessage(WM_CLOSE);
}

```

```
void CScreensavealbumDlg::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
```

```

{
 PostMessage(WM_CLOSE);
}

```

```
void CScreensavealbumDlg::OnLButtonDown(UINT nFlags, CPoint point)
```

```

{
 PostMessage(WM_CLOSE);
}

```

```
void CScreensavealbumDlg::OnLButtonUp(UINT nFlags, CPoint point)
```

```

{
 PostMessage(WM_CLOSE);
}

```

```
void CScreensavealbumDlg::OnRButtonDown(UINT nFlags, CPoint point)
```

```

{
 PostMessage(WM_CLOSE);
}

```

```
void CScreensavealbumDlg::OnRButtonUp(UINT nFlags, CPoint point)
```

```

{
 PostMessage(WM_CLOSE);
}

```

## 举一反三

根据本实例, 读者可以:

- 开发具有随机变换效果的电子相册屏幕保护程序。

## 实例 166 产品宣传屏幕保护程序

本实例是一个人性化的实例

实例位置: 光盘\mingrisoft\04\166

## 实例说明

由于触摸屏在信息查询上以触摸的形式出现,被各企事业单位所广泛采用。如医院、邮电通信业、工商局、税务局、政府机构、宾馆、展销会、航空公司售票处、国际游轮售票处、银行、证券交易所、图书馆等。如果能在触摸屏查询程序不被人使用的情况下,为商家作一些产品宣传,则商家将受益匪浅。本实例将实现制作产品宣传屏幕保护程序,如图 4.14 所示。

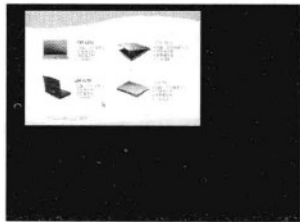


图 4.14 产品宣传屏幕保护程序

## 技术要点

本实例是通过全屏显示的,在背景色为黑色的对话框上绘制图片来实现屏幕保护程序。屏幕保护是扩展名为 src 的应用程序,并且屏幕保护程序都存放在系统 system32 文件夹下。屏幕保护应该在用户触发鼠标和键盘消息时关闭,并在运行的时候隐藏鼠标。本实例使用 ShowCursor 函数来隐藏鼠标,如果要对鼠标移动进行处理还需要在隐藏鼠标的同时记录鼠标的位置,当程序触发 WM\_MOUSEMOVE 消息时将鼠标当前的位置和鼠标先前的位置进行比较,如果移动超出了一定的范围就停止屏幕保护。

## 实现过程

- (1) 新建一个名为 ScrnSaverProduct 的对话框 MFC 工程。
- (2) 将对话框的 Border 属性设置为 None。
- (3) 在头文件定义函数及变量声明,代码如下:

```
CPoint curpt;
int x;
int y;
int idirect;
int a;
int b;
void DrawBitmap(CDC &dc, int nIndexBit);
```

- (4) 在 OnInitDialog 函数中将设置屏幕大小,并对一些变量进行初始化,代码如下:

```
BOOL CScrnSaverProductDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...
 ShowCursor(false);
 CRect rc(0,0,GetSystemMetrics(SM_CXSCREEN),GetSystemMetrics(SM_CYSCREEN));
 ::GetCursorPos(&curpt); //获取鼠标位置
 this->MoveWindow(rc); //移动窗口
 SetTimer(1,500,NULL);
 x=0;y=0;
 a=0;b=0;
 idirect=0;
 return TRUE;
}
```

- (5) 添加对 WM\_MOUSEMOVE 消息的实现函数,实现移动鼠标关闭程序,代码如下:

```
void CScrnSaverProductDlg::OnMouseMove(UINT nFlags, CPoint point)
{
 int moveptx=point.x-curpt.x;
 int movepty=point.y-curpt.y;
 if(moveptx+movepty>3)
 PostMessage(WM_CLOSE); //发送窗口关闭消息
 CDialog::OnMouseMove(nFlags, point);
}
```



(6) 在 OnPaint 函数中实现黑色背景的绘制, 代码如下:

```
void CScmSaverProductDlg::OnPaint()
{
 CPaintDC dc(this);
 CBrush brush(0,0,0);
 CRect rect;
 GetClientRect(rect); // 获取窗口客户区
 dc.FillRect(&rect,&brush);
}
```

(7) 关闭窗口时关闭程序, 代码如下:

```
void CScmSaverProductDlg::PostNcDestroy()
{
 KillTimer(1);
 CDialog::PostNcDestroy();
}
```

(8) 设置定时器, 在定时器内通过自定义函数绘制图片, 代码如下:

```
void CScmSaverProductDlg::OnTimer(UINT nIDEvent)
{
 KillTimer(1);
 CClientDC dc(this);
 static nIndexBit=0;
 if(nIndexBit>3)
 nIndexBit=0;
 DrawBitmap(dc, nIndexBit++); // 绘制图片
 SetTimer(1,500,NULL);
 CDialog::OnTimer(nIDEvent);
}
```

(9) 自定义函数 DrawBitmap, 实现图片的绘制, 代码如下:

```
void CScmSaverProductDlg::DrawBitmap(CDC &dc, int nIndexBit)
{
 CDC dcmem;
 dcmem.CreateCompatibleDC(&dc);
 CBitmap m_Bitmap;
 m_Bitmap.LoadBitmap(IDB_BITMAP1+nIndexBit);
 dcmem.SelectObject(m_Bitmap);
 BITMAP bmp;

 GetObject(m_Bitmap, sizeof(bmp), &bmp);
 int iscreenx=GetSystemMetrics(SM_CXSCREEN); // 获取屏幕宽度
 int iscreeny=GetSystemMetrics(SM_CYSCREEN); // 获取屏幕高度
 if(a>iscreenx/2)a=0;
 if(b>iscreeny/2)b=0;
 if(idirect>1)idirect=0;
 switch(idirect)
 {
 case 0:
 for(x=bmp.bmWidth;x>0;x--)
 {
 dc.BitBlt(a,b,bmp.bmWidth,bmp.bmHeight,&dcmem,x,y,SRCCOPY); // 将图片绘制在窗口上
 }
 break;
 case 1:
 for(y=bmp.bmHeight;y>0;y--)
 {
 dc.BitBlt(a,b,bmp.bmWidth,bmp.bmHeight,&dcmem,x,y,SRCCOPY);
 }
 break;
 }
 indirect++;
 a+=60;
 b+=20;
 Sleep(2000);
 dc.BitBlt(0,0,iscreenx,iscreeny,&dcmem,0,0,BLACKNESS);
 dcmem.DeleteDC();
}
```

### 举一反三

根据本实例, 读者可以:

- 开发 VCD 播放屏幕保护程序。

## 实例 167 滚动字幕屏幕保护程序

本实例是一个人性化的实例

实例位置: 光盘\mingrisoft\04\167

## 实例说明

屏幕保护程序是当系统在指定的时间内没有发生任何用户操作时, 自动启动的一个程序, 通过该程序的启动起到对屏幕信息的保护作用, 如图 4.15 所示。

## 技术要点

屏幕保护就是扩展名为 .src 的应用程序, 将应用程序扩展名 .exe 改为 .src 后放在 system32 文件夹下, 系统就将应用程序识别为屏幕保护程序。所以制作一个屏幕保护程序就是制作一个应用程序。

图 4.15 滚动字幕  
屏幕保护程序

## 实现过程

- (1) 新建一个基于对话框的 MFC 工程。
- (2) 设置窗体的 Border 属性为 None。
- (3) 主要程序代码。

```
CPoint curpt; //鼠标在初始化时所在位置
int x,y; //输出文字的左顶点位置
int iscreenx;
int iscreeny;
//在初始化时将程序设置为屏幕大小
BOOL CTextScreenSaveDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//代码省略
 ShowCursor(false); //隐藏鼠标
 iscreenx=GetSystemMetrics(SM_CXSCREEN);
 iscreeny=GetSystemMetrics(SM_CYSCREEN);
 CRect rc(0,0,iscreenx,iscreeny);
 ::GetCursorPos(&curpt); //获取鼠标位置
 this->MoveWindow(rc);
 x=0;y=0;
 AfxBeginThread(thread,this); //启动一个线程来改变字符输出的位置
 return TRUE;
}
//将对话框背景设置为黑色, 创建一个新字体并输出
void CTextScreenSaveDlg::OnPaint()
{
 if (IsIconic())
 {
 ...//代码省略
 }
 else
 {
 CPaintDC dc(this);
 CBrush brush(RGB(0,0,0));
 CRect rect;
 GetClientRect(rect);
 dc.FillRect(&rect,&brush);
 CFont font;
 font.CreateFont(30,20,10,10,FW_NORMAL,FALSE,FALSE,0,
 ANSI_CHARSET,OUT_DEFAULT_PRECIS,
 CLIP_DEFAULT_PRECIS,
 DEFAULT_QUALITY,DEFAULT_PITCH|FF_SWISS,""); //创建字体
 dc.SelectObject(font);
 dc.SetTextColor(RGB(0,255,255)); //设置字体的颜色
 dc.SetBkMode(TRANSPARENT); //设置字体的透明模式
 TEXTMETRIC tm;
 ::GetTextMetrics(dc.GetSafeHdc(),&tm);
```

```
rect.SetRect(x,y,tm.tmMaxCharWidth*10,tm.tmHeight+y);
dc.DrawText("mingrisoft",&rect,DT_LEFT);//向设备上下文中输出字符
CDialog::OnPaint();
}

//线程的实现函数
static UINT thread(LPVOID pParam)
{
 CTextScreenSaveDlg *p=(CTextScreenSaveDlg*)pParam;

 CDC *pDC=p->GetDC();
 while(1)
 {
 p->x+=10;
 if(p->x>p->iscreenx)
 {
 p->x=0;
 p->y+=20;
 }
 if(p->y>p->iscreeny)p->y=0;
 Sleep(10);
 p->Invalidate(FALSE);//重绘
 }
 return 0;
}

//当移动鼠标时，退出屏幕保护
void CTextScreenSaveDlg::OnMouseMove(UINT nFlags, CPoint point)
{
 int moveptx=point.x-curpt.x;
 int movepty=point.y-curpt.y;
 if(abs(moveptx)+abs(movepty)>3)
 PostMessage(WM_CLOSE);
 CDialog::OnMouseMove(nFlags, point);
}

//当按下键盘或鼠标左右键后，退出屏幕保护
BOOL CTextScreenSaveDlg::PreTranslateMessage(MSG* pMsg)
{
 if(pMsg->message==WM_KEYDOWN)
 PostMessage(WM_CLOSE);
 if(pMsg->message==WM_LBUTTONDOWN)
 PostMessage(WM_CLOSE);
 if(pMsg->message==WM_RBUTTONDOWN)
 PostMessage(WM_CLOSE);
 return CDialog::PreTranslateMessage(pMsg);
}
```

### 举一反三

根据本实例，读者可以：

- 开发 VCD 播放屏幕保护程序。

## 4.5 DirectShow 程序设计

DirectShow 是 DirectX 开发包中关于流媒体处理的一个开发包，这个开发包可以进行音频和视频的捕捉，可以开发 DVD 应用程序，还可以开发数字的 TV 应用程序。本节通过几个实例介绍 DirectShow 程序设计方法。

### 实例 168 音频捕捉

这是一个可以提高基础性能的实例

实例位置：光盘\mingrisoft\04\168

### 实例说明

许多读者都使用过 Windows 的录音机工具，该工具能够录制声卡播放的声音。那么如何在

程序中实现录音功能呢？本例实现了声卡的音频捕捉功能。运行程序，结果如图 4.16 所示。

### 技术要点

本例中需要实现一个音频捕捉的应用程序，将捕捉到的声卡信息写入到文件中。首先需要构建一个音频捕捉过滤图表。根据需要，用户可以采取 3 种方案。

- (1) 将声音保存到 AVI (只有声音) 文件中。
- (2) 将声音保存到 WAV 文件中。
- (3) 将声音保存到 ASF 文件中。

本例笔者采用第二种方案，将声音保存到 WAV 文件中。需要说明的是 DirectX 并没有提供写入 WAV 文件中的 Filter (过滤器)，但是在 DirectX 安装目录下的 Samples\Multimedia\DirectShow\Filters\WavDest 子目录中提供了一个例子，通过注册该实例可以获得一个写入 WAV 文件的 Filter。方法很简单，首先编译并运行该实例，然后选择 Tools/Register Control 菜单项进行注册就可以了。

下面笔者介绍利用 DirectX 提供的 GraphEdit 工具设计音频捕捉过滤图表。

(1) 单击操作系统“开始”菜单，选择 GraphEdit 工具，如图 4.17 所示，打开 GraphEdit 窗口，如图 4.18 所示。

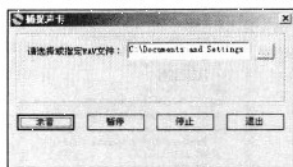


图 4.16 音频捕捉

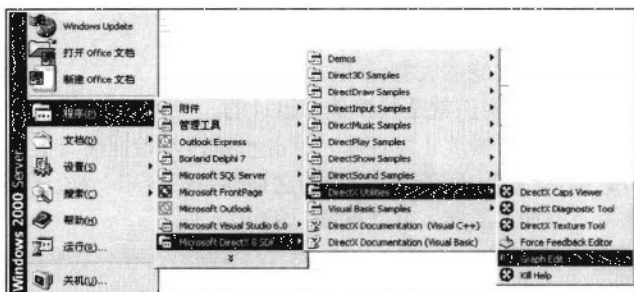


图 4.17 从系统菜单选择 GraphEdit 工具

(2) 选择 Graph\Insert Filters 菜单项，打开插入过滤器窗口，在 Audio Capture Sources 节点下会列出音频捕捉过滤器，每一项对应于系统中的一个声卡。选中一个过滤器，例如 Realtek AC97 Audio，如图 4.19 所示，单击“Insert Filter”按钮向音频捕捉过滤图表中添加音频捕捉过滤器。

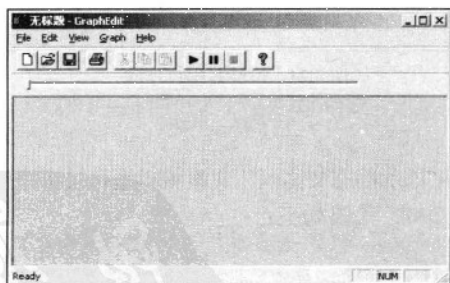


图 4.18 GraphEdit 窗口

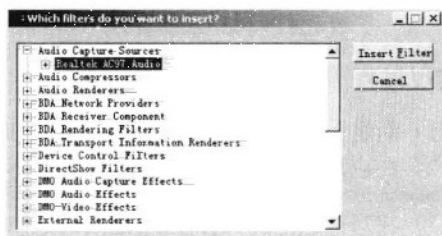


图 4.19 添加音频捕捉过滤器

(3) 在 DirectShow Filters 节点下选择 Wav Dest 节点，如图 4.20 所示，单击“Insert Filter”按钮添加 Wav Dest 过滤器，用于向 WAV 文件中写入音频流。

**注意：**如果之前没有注册 Samples\Multimedia\DirectShow\Filters\WavDest 目录的实例，列表中则不会有 Wav Dest 节点。



(4) 在 DirectShow Filters 节点下选择 File writer 节点, 如图 4.21 所示, 单击 “Insert Filter” 按钮添加 File Writer 过滤器, 用于向文件中写入数据。

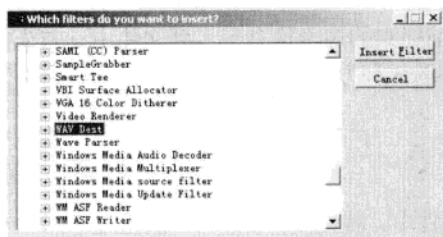


图 4.20 添加 Wav Dest 过滤器

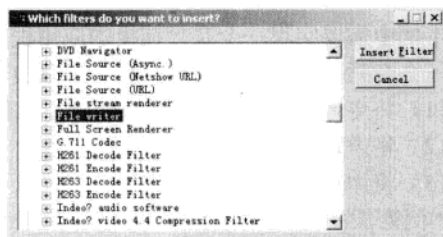


图 4.21 添加 File Writer 过滤器

(5) 此时, 系统会弹出一个窗口, 让用户为 File Writer 选择一个输出文件, 这里输入 temp2.wav, 如图 4.22 所示。

(6) 单击 “打开” 按钮完成设置, 然后单击 “Close” 按钮关闭插入过滤器窗口, 返回到 Graph Edit 主窗口, 发现窗口中多了 3 个选项 (添加的过滤器 Filter), 如图 4.23 所示。

(7) 利用鼠标将 3 个过滤器的端口连接起来, 至此就完成了音频过滤图表的设计, 如图 4.24 所示。

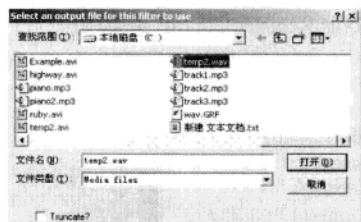


图 4.22 设置输出文件

有些读者可能会问: 音频过滤图表有何用处? 它体现了音频捕捉的原理和过程。本节实例就是根据该图设计的。读者可以进行测试。在计算机中播放一段音乐, 单击 ▶ 按钮开始捕捉, 在适当的时候单击 ■ 按钮停止捕捉, 播放 File Writer 指定的文件, 发现与之前播放的音乐一样。设计好过滤图表之后, 程序中只要按照过滤图表添加过滤器, 并连接端口就可以了。

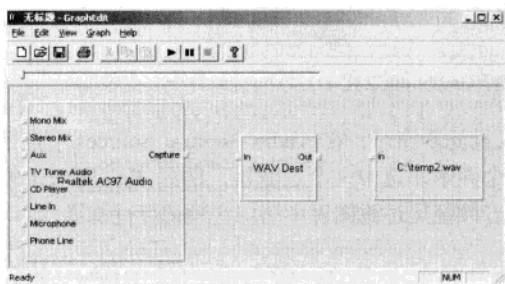


图 4.23 GraphEdit 主窗口

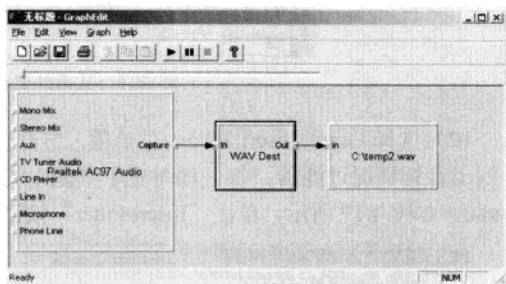


图 4.24 过滤器端口连线

## 实现过程

- (1) 新建一个基于对话框的应用程序。在对话框中添加静态文本框、文本编辑框和按钮控件。
- (2) 在应用程序的 InitInstance 方法中初始化 COM, 代码如下:  
CoInitialize(NULL);
- (3) 在对话框的头文件中引用 “dshow.h” 头文件。
- (4) 在对话框的头文件中定义如下成员变量。代码如下:  
IMediaControl\* pMediaControl; //媒体控制  
IMoniker\* pMoniker;  
IBaseFilter\* pSrc, \*pWaveDest, \*pWriter; //定义过滤器Filter  
IFileSinkFilter2\* pSink; //将媒体流写入文件  
IGraphBuilder\* pGraph; //构建步骤图表  
BOOL m\_IsPause; //是否暂停  
BOOL m\_IsRecorded; //是否进行了录音
- (5) 添加一个自定义函数, 用于查找 Filter 的端口, 代码如下:

```
IPin* CWavDlg::FindPin(IBaseFilter *pFilter, PIN_DIRECTION dir)
{
 IEnumPins* pEnumPins;
 IPin* pOutpin;
 PIN_DIRECTION pDir;
 pFilter->EnumPins(&pEnumPins);

 while (pEnumPins->Next(1,&pOutpin,NULL)==S_OK)
 {
 pOutpin->QueryDirection(&pDir);
 if (pDir==dir)
 {
 return pOutpin;
 }
 }
 return 0;
}
```

(6) 处理“录音”按钮的单击事件，开始录音，代码如下：

```
void CWavDlg::OnOK()
{
 CString str;
 m_WavFile.GetWindowText(str);
 if (str.IsEmpty())
 {
 MessageBox("请选择或输入文件");
 return;
 }

 ICaptureGraphBuilder2 * pBuilder = NULL;
 pGraph = NULL;
 pMediaControl = NULL;
 //创建ICaptureGraphBuilder2
 CoCreateInstance(CLSID_CaptureGraphBuilder2,0,
 CLSCTX_INPROC_SERVER, IID_ICaptureGraphBuilder2, (void**)&pBuilder);
 //创建pGraph
 CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
 IID_IGraphBuilder, (void**)&pGraph);

 //设置图表
 pBuilder->SetFiltergraph(pGraph);

 //创建媒体控制对象pMediaControl
 pGraph->QueryInterface(IID_IMediaControl, (void**)&pMediaControl);

 /*****列举并选择一个音频过滤器*****/

 ICreateDevEnum *pDevEnum = NULL;
 CoCreateInstance(CLSID_SystemDeviceEnum, NULL, CLSCTX_INPROC,
 IID_ICreateDevEnum, (void**)&pDevEnum);

 IEnumMoniker *pClassEnum = NULL;
 pDevEnum->CreateClassEnumerator(CLSID_AudioInputDeviceCategory, &pClassEnum, 0);
 ULONG cFetched;
 if (pClassEnum->Next(1, &pMoniker, &cFetched) == S_OK)
 {
 pMoniker->BindToObject(0, 0, IID_IBaseFilter, (void**)&pSrc);
 pMoniker->Release();
 }
 pClassEnum->Release();
 /*****/

 //创建WavDest过滤器Filter
 CoCreateInstance(CLSID_WavDest, NULL, CLSCTX_ALL,
 IID_IBaseFilter, (void**)&pWaveDest);

 //创建File Writer过滤器Filter
 CoCreateInstance(CLSID_FileWriter, NULL, CLSCTX_ALL,
 IID_IBaseFilter, (void**)&pWriter);

 //向图表中添加过滤器
 pGraph->AddFilter(pSrc,L"Wav");
 pGraph->AddFilter(pWaveDest,L"WavDest");
 pGraph->AddFilter(pWriter,L"FileWriter");

 //创建IFileSinkFilter2对象
 pWriter->QueryInterface(IID_IFileSinkFilter2, (void**)&pSink);

 //设置IFileSinkFilter2对象输出的媒体文件
 pSink->SetFileName(str.AllocSysString(), NULL);

 //查找音频过滤器的输出端口
```

```
IPin* pOutpin = FindPin(pSrc,PINDIR_OUTPUT);
IPin* pInpin,*pOut;
//查找Wav Dest的输出端口
pOut= FindPin(pWaveDest,PINDIR_OUTPUT);
//查找Wav Dest的输入端口
pInpin = FindPin(pWaveDest,PINDIR_INPUT);
//查找File Writer的输入端口
IPin* pInpin1= FindPin(pWriter,PINDIR_INPUT);
//连接音频捕捉过滤器和WavDest过滤器端口
pGraph->ConnectDirect(pOutpin,pInpin,NULL);
//连接WavDest过滤器和FileWriter过滤器端口
pGraph->ConnectDirect(pOut,pInpin1,NULL);
pMediaControl->Run();//运行过滤器Filter
m_IsRecorded = TRUE;
}
```

### 举一反三

根据本实例的设计思路和一些技术要点，读者还可以：

- 开发视频捕捉程序。

## 实例 169 音频压缩

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\04\169

### 实例说明

音频压缩是指将通过音频流获取的音频数据进行压缩处理，使用 acm 函数进行音频压缩时，用户需要了解 Wave 文件格式，并熟悉使用 mmio 函数。下面结合具体实例介绍使用 acm 函数进行音频压缩。运行程序，如图 4.25 所示。

### 技术要点

Wave 文件是常用的声波文件格式之一，与 AVI 文件格式相同，Wave 文件也是采用 RIFF 文件格式。Wave 文件主要有两种，单声道和多声道。单声道的采样率为 11.025kHz，采样值是 8 bit，双声道的采样率为 44.1kHz，采样值是 16 bit。这里的采样率是指声音信号在从模拟信号向数字信号的转换过程中，单位时间内的采样次数。采样值是指每一次采样周期内声音模拟信号的积分值。

Wave 文件包含文件头和数据两部分，其中，数据部分就是音频的实际数据，在此不作讨论，下面介绍 Wave 文件的文件头。图 4.26 详细描述了 Wave 文件格式。

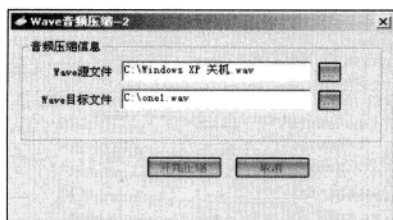


图 4.25 音频压缩

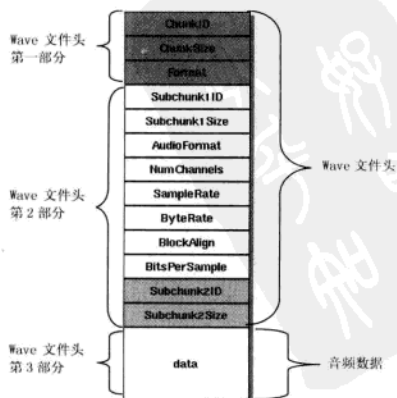


图 4.26 Wave 文件格式

从图 4.26 中可以看出, Wave 文件头主要包含 3 部分, 第一部分是一个 RIFF LIST 数据结构, 定义如下:

```
typedef struct _rifflist {
 FOURCC fcc;
 DWORD cb;
 FOURCC fccListType;
} RIFFLIST, * LPRIFFLIST;
```

其中, fcc 的值为 FCC('RIFF'), 也就是 Wave 文件的前 4 个字符为“RIFF”。cb 表示其后的数据块的大小。fccListType 的值为 FCC('WAVE'), 表示 Wave 类型。

Wave 文件头的第二部分包含两个数据结构, 分别为 RIFFCHUNK 和 WAVEFORMATEX。

其中 RIFFCHUNK 结构定义如下:

```
typedef struct _riffchunk {
 FOURCC fcc;
 DWORD cb;
} RIFFCHUNK, * LPRIFFCHUNK;
```

其中, fcc 为 FCC('fmt'), 表示格式数据块 fmt, cb 表示其后的数据块大小。

WAVEFORMATEX 结构定义如下:

```
typedef struct waveformat_extended_tag {
 WORD wFormatTag;
 WORD nChannels;
 DWORD nSamplesPerSec;
 DWORD nAvgBytesPerSec;
 WORD nBlockAlign;
 WORD wBitsPerSample;
 WORD cbSize;
} WAVEFORMATEX;
```

其中, wFormatTag 表示音频格式, nChannels 表示声道数量, nSamplesPerSec 表示采样率, nAvgBytesPerSec 表示平均数据速率, nBlockAlign 表示数据对齐, wBitsPerSample 表示音频采样大小, cbSize 表示 WAVEFORMATEX 结构之后数据的大小。

## 实现过程

- (1) 创建一个基于对话框的应用程序, 向对话框中添加按钮、编辑框、静态文本等控件。
- (2) 在对话框的头文件中引用相应的头文件和库文件。代码如下:

```
#include <math.h>
#pragma comment (lib, "msacm32.lib")
#pragma comment (lib, "winmm.lib")
#include "vfw.h"
#pragma comment (lib, "vfw32")
```

- (3) 定义 Wave 文件格式中数据块的结构。代码如下:

```
//定义Wave文件格式中数据块结构
typedef struct _rifflist
{
 FOURCC fcc;
 DWORD cb;
 FOURCC fccListType;
} RIFFLIST, * LPRIFFLIST;
typedef struct _riffchunk
{
 FOURCC fcc;
 DWORD cb;
} RIFFCHUNK, * LPRIFFCHUNK;
typedef struct {
 WORD wFormatTag;
 WORD nChannels;
 DWORD nSamplesPerSec;
 DWORD nAvgBytesPerSec;
 WORD nBlockAlign;
 WORD wBitsPerSample;
} WAVEFORMATEX1;
```

- (4) 从 MSDN 中复制 wave.h、wave.c 文件, 将其添加到工程中。在这两个文件中提供了操作 wave 文件的简单方法, 实际上是对 mmio 函数的封装。



(5) 在 wave.c 文件中修改 WaveWriteFile 函数, 将 “\*((BYTE\*)pmmioinfoOut->pchNext) = \*((BYTE\*)pbSrc+cT);” 代码修改为:

```
((BYTE)pmmioinfoOut->pchNext) = *((BYTE*)pbSrc+cT);
(BYTE*) pmmioinfoOut->pchNext++;
```

因为在 Visual C++ 环境中, 源代码无法编译。

(6) 定义如下全局变量, 记录压缩后的音频格式及采样信息等。

```
WAVEFORMATEX dstPCM; //目标音频格式
DWORD dstSamples; //目标采样数量
HACMDRIVERID hAcm; //ACM驱动程序句柄
DWORD dstBytes; //压缩后的音频数据大小
```

(7) 添加 FormatEnumProc 全局函数, 获取指定的目标音频格式。代码如下:

```
//选择目标音频压缩格式
BOOL CALLBACK FormatEnumProc(HACMDRIVERID hadid, LPACMFORMATDETAILS pafid,
 DWORD dwInstance, DWORD fdwSupport)
{
 if(pafid->dwFormatTag==WAVE_FORMAT_PCM)
 {
 DWORD sample=pafid->pwfx->nAvgBytesPerSec;
 if((sample==44100))
 {
 dstPCM =*(WAVEFORMATEX*)pafid->pwfx;
 hAcm=hadid;
 return FALSE;
 }
 }
 return TRUE;
}
```

(8) 添加 CodecsEnum 函数, 列举音频驱动程序, 列举支持的音频格式。代码如下:

```
BOOL CALLBACK CodecsEnum(HACMDRIVERID hAdid, DWORD dwInstance,DWORD dwSupport)
{
 DWORD dwSize = 0;

 ACMDRIVERDETAILS acm;
 acm.cbStruct = sizeof(acm);
 MMRESULT mmr = acmDriverDetails(hAdid, &acm, 0);

 HACMDRIVER had = NULL;
 mmr = acmDriverOpen(&had,hAdid,0); //打开驱动程序
 if (mmr)
 {
 return FALSE;
 }
 else
 {
 mmr = acmMetrics((HACMOBJ)had, ACM_METRIC_MAX_SIZE_FORMAT, &dwSize);
 if (dwSize < sizeof(WAVEFORMATEX)) dwSize = sizeof(WAVEFORMATEX);
 WAVEFORMATEX* pWaveForm = (WAVEFORMATEX*) malloc(dwSize);
 memset(pWaveForm,0,dwSize);
 pWaveForm->cbSize = LOWORD(dwSize)- sizeof(WAVEFORMATEX);
 pWaveForm->wFormatTag = WAVE_FORMAT_UNKNOWN;
 ACMFORMATDETAILS AcForm;
 memset(&AcForm, 0, sizeof(AcForm));
 AcForm.cbStruct = sizeof(AcForm);
 AcForm.pwfx = pWaveForm;
 AcForm.cbwfx = dwSize;
 AcForm.dwFormatTag = WAVE_FORMAT_UNKNOWN;
 mmr = acmFormatEnum(had, &AcForm, FormatEnumProc, 0, 0);
 if (mmr||hAcm)
 {
 return FALSE;
 }
 free(pWaveForm);
 acmDriverClose(had, 0);
 }
 return TRUE;
}
```

(9) 向对话框中添加 OnConvert 方法, 用于将源音频数据转换为目标音频数据并返回。代码如下:

```

BYTE* CAudioCompressDlg::OnConvert(BYTE* pSrcData,DWORD datasize)
{
 //数据块的调整数
 m_wfSrc.nBlockAlign = m_wfSrc.nChannels *m_wfSrc.wBitsPerSample /8;

 MMRESULT mmr;
 HACMSTREAM hstr = NULL;
 mmr = acmStreamOpen(&hstr,
 NULL,
 (WAVEFORMATEX*)&m_wfSrc,
 (WAVEFORMATEX*)&dstPCM,
 NULL,
 0,
 ACM_STREAMOPENF_NONREALTIME);

 if (mmr) {
 AfxMessageBox("转换失败!");
 return NULL;
 }
 ACMSTREAMHEADER strhdr;
 memset(&strhdr, 0, sizeof(strhdr));
 strhdr.cbStruct = sizeof(strhdr);
 strhdr.pbSrc = pSrcData;
 strhdr.cbSrcLength = datasize;
 //采样数量
 dstSamples = datasize / (m_wfSrc.nChannels* m_wfSrc.wBitsPerSample/8);

 dstBytes=datasize ;

 BYTE* pDstData = new BYTE[dstBytes];
 strhdr.cbDstLength = dstBytes;
 strhdr.pbDst = pDstData;
 mmr = acmStreamPrepareHeader(hstr, &strhdr, 0);
 mmr = acmStreamConvert(hstr, &strhdr, 0);
 dstBytes = strhdr.cbDstLengthUsed;
 if (mmr)
 {
 AfxMessageBox("转换失败!");
 return NULL ;
 }

 acmStreamClose(hstr, 0);
 return pDstData;
}

```

(10) 处理“开始压缩”按钮的单击事件，对源音频文件进行压缩，生成目标文件。代码如下：

void CAudioCompressDlg::OnCompress()

```

{
 CString srcFile;
 m_SrcFile.GetWindowText(srcFile);
 CString dstFile;
 m_DstFile.GetWindowText(dstFile);
 //列举驱动器
 acmDriverEnum(CodecsEnum,0,0);

 CFile fp;
 fp.Open(srcFile ,CFile::modeRead);
 DWORD dwDstBytes=fp.GetLength();

 RIFFLIST riff;

 fp.Read(&riff,sizeof(RIFFLIST));
 RIFFCHUNK riffchunk;
 fp.Read(&riffchunk,sizeof(RIFFCHUNK));

 fp.Read(&m_wfSrc,sizeof(WAVEFORMATEX1));

 RIFFCHUNK riffData;
 fp.Read(&riffData,sizeof(RIFFLIST));

 BYTE* pSrcData=new byte [riffData.cb];
 fp.Read(pSrcData, riffData.cb);

 fp.Close();
}

```

```

HMMIO hmmio;
MMCKINFO pck,riffinfo;

UINT factsize;

MMIOINFO mmio;

BYTE* pDstData = OnConvert(pSrcData,riffData.cb);

WaveSaveFile(dstFile.GetBuffer(0),dstBytes,
dstSamples,(WAVEFORMATEX*)&dstPCM,pDstData);
delete [] pSrcData;
delete [] pDstData;
}

```

### 举一反三

根据本实例，读者可以：

- 使用 DirectShow 进行音频压缩。

## 实例 170 视频捕捉

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\04\170

### 实例说明

几乎每个视频应用程序都包含视频捕捉功能。视频捕捉能够将摄像头捕捉的信息保存至磁盘中。在需要的时候，可以像播放影片一样播放视频文件。本例实现的视频捕捉的功能。运行程序，如图 4.27 所示。

### 技术要点

在设计视频捕捉时，经常将摄像头捕捉的信息保存为 AVI 文件。使用 DirectX 开发包可以很容易实现视频捕捉。首先利用 GraphEdit 工具设计过滤器图表，具体步骤可参考上一实例中的“技术要点”，过滤器图表最终如图 4.28 所示。

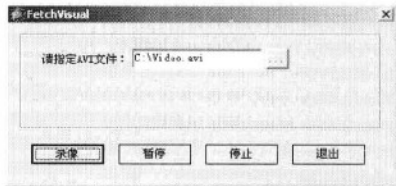


图 4.27 视频捕捉

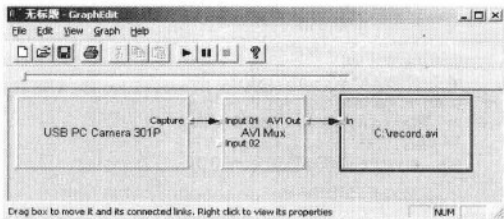


图 4.28 过滤器图表

然后按照过滤图表添加过滤器，连接端口。详细代码可参见实现过程。

### 实现过程

- (1) 新建一个基于对话框的应用程序。在对话框中添加静态文本、文本编辑和按钮控件。
- (2) 在应用程序的 InitInstance 方法中初始化 COM，代码如下：  
CoInitialize(NULL);
- (3) 在对话框的头文件中引用“dshow.h”头文件。
- (4) 在对话框的头文件中定义如下成员变量：  
IMediaControl \*pMediaControl; //媒体控制  
IMoniker \*pMoniker;

```
IBaseFilter *pSrc,*pMux,*pWriter;
IFileSinkFilter2 *pSink;
IGraphBuilder *pGraph;
BOOL m_IsPause; //是否暂停
BOOL m_IsRecorded; //是否进行了录音
```

(5) 添加一个自定义函数；用于查找Filter的端口，代码如下：

```
IPin* CFetchVisualDlg::FindPin(IBaseFilter *pFilter, PIN_DIRECTION dir)
{
 IEnumPins* pEnumPins;
 IPin* pOutpin;
 PIN_DIRECTION pDir;
 pFilter->EnumPins(&pEnumPins);

 while (pEnumPins->Next(1,&pOutpin,NULL)==S_OK)
 {
 pOutpin->QueryDirection(&pDir);
 if (pDir==dir)
 {
 return pOutpin;
 }
 }
 return 0;
}
```

(6) 处理“录像”按钮的单击事件，将摄像头捕捉的信息保存到磁盘中，代码如下：

```
void CFetchVisualDlg::OnOK()
{
 CString str;
 m_File.GetWindowText(str);
 if (str.IsEmpty())
 {
 MessageBox("请选择或输入文件");
 return;
 }

 ICaptureGraphBuilder2 *pBuilder = NULL;

 pGraph = NULL;

 pMediaControl = NULL;

 /*****列举视频设备*****/
 ICreateDevEnum *pDevEnum = NULL;

 CoCreateInstance(CLSID_SystemDeviceEnum, NULL, CLSCTX_INPROC,
 IID_ICreateDevEnum, (void **)&pDevEnum);

 IEnumMoniker *pClassEnum = NULL;

 pDevEnum->CreateClassEnumerator(CLSID_VideoInputDeviceCategory, &pClassEnum, 0);
 ULONG cFetched;
 if (pClassEnum->Next(1, &pMoniker, &cFetched) == S_OK)
 {
 pMoniker->BindToObject(0, 0, IID_IBaseFilter, (void **)&pSrc;
 pMoniker->Release();
 }
 pClassEnum->Release();
 /*****列举视频设备*****/

 /*****创建过滤器*****/

 CoCreateInstance(CLSID_CaptureGraphBuilder2, 0, CLSCTX_INPROC_SERVER,
 IID_ICaptureGraphBuilder2, (void **)&pBuilder);

 CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
 IID_IGraphBuilder, (void **)&pGraph);

 pBuilder->SetFiltergraph(pGraph);

 pGraph->QueryInterface(IID_IMediaControl, (void **)&pMediaControl);

 pGraph->AddFilter(pSrc, L"avi");

 CoCreateInstance(CLSID_AviDest, NULL, CLSCTX_ALL,
```



```

 IID_IBaseFilter,(void*)&pMux);
pGraph->AddFilter(pMux,L"Mux");

CoCreateInstance(CLSID_FileWriter, NULL, CLSCTX_ALL,
 IID_IBaseFilter, (void*)&pWriter);
pGraph->AddFilter(pWriter,L"Writer");

pWriter->QueryInterface(IID_IFileSinkFilter2,(void*)&pSink);
pSink->SetFileName(str.AllocSysString(),NULL);

/*****创建过滤器*****/
/*****连接端口*****/

IPin* pOutpin = FindPin(pSrc,PINDIR_OUTPUT); //pSrc的输出端口
IPin* pInpin,*pOut; //pMux的输入/输出端口
pInpin = FindPin(pMux,PINDIR_INPUT);
pOut= FindPin(pMux,PINDIR_OUTPUT);
IPin* pInpin1= FindPin(pWriter,PINDIR_INPUT); //pWriter的输入端口

//连接端口
HRESULT result ;
result = pGraph->ConnectDirect(pOutpin,pInpin,NULL);
result = pGraph->ConnectDirect(pOut,pInpin1,NULL);

pMediaControl->Run();
m_IsRecorded = TRUE;
/*****连接端口*****/
}

```

### 举一反三

根据本实例，读者可以：

- 开发数码相机应用程序。

## 实例 171 视频压缩

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\04\171

### 实例说明

进行数据压缩可以使用 AVI 函数和 ICM 函数。首先利用 AVI 函数打开 AVI 文件，获取流信息，打开视频帧，读取帧数据，然后利用 ICM 函数选择一个压缩器，对帧数据进行压缩，最后将压缩的数据写入视频流中。运行程序，如图 4.29 所示。

### 技术要点

在进行视频数据压缩处理时读者需要对 AVI 函数及 ICM 函数的使用有所了解。下面将对这两类函数分别进行介绍。

#### 1. AVI 函数

(1) AVIFileInit 函数。该函数用于初始化 AVIFile 函数库，在使用 AVI 函数之前，首先需要调用该函数进行初始化。语法如下：

```
STDAPI_(VOID) AVIFileInit(VOID);
```

(2) AVIFileExit 函数。该函数的作用与 AVIFileInit 函数相反，它用于退出 AVIFile 函数库。语法如下：

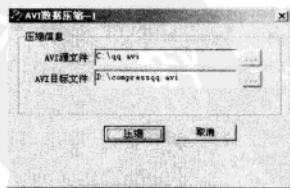


图 4.29 视频压缩

STDAPI\_(VOID) AVIFileExit(VOID);

(3) AVIFileOpen 函数。该函数用于打开一个 AVI 文件，并返回一个用于访问文件的接口地址。语法如下：

STDAPI AVIFileOpen(PAVIFILE \* ppfile, LPCTSTR szFile, UINT mode, CLSID \* pclsidHandler);

参数说明：

- ppfile：表示文件接口指针的地址。
- szFile：表示文件名称。
- mode：表示打开文件时的访问模式，可选值如表 4.6 所示。

表 4.6 mode 可选值

| mode 可选值            | 描 述                                      |
|---------------------|------------------------------------------|
| OF_CREATE           | 如果文件不存在，则创建文件，如果文件已经存在，则删除文件内容           |
| OF_SHARE_DENY_NONE  | 以不排它的形式打开文件，其他进程可以访问 AVIFileOpen 函数打开的文件 |
| OF_SHARE_DENY_READ  | 以不排它的形式打开文件，其他进程能够以写形式访问文件               |
| OF_SHARE_DENY_WRITE | 以不排它的形式打开文件，其他进程能够以读形式访问文件               |
| OF_SHARE_EXCLUSIVE  | 以排它的形式打开文件，其他进程不能访问该文件                   |
| OF_READ             | 以只读的形式打开文件                               |
| OF_READWRITE        | 以读写形式打开文件                                |
| OF_WRITE            | 以写的形式打开文件                                |

- pclsidHandler：表示标准或用户自定义类的标识符的地址，可以为 NULL。

(4) AVIFileGetStream 函数。该函数用于获得与 AVI 文件指针关联的流信息。语法如下：

STDAPI AVIFileGetStream(PAVIFILE pfile, PAVISTREAM \* ppavi, DWORD fccType, LONG lParam);

参数说明：

- pfile：表示打开的 AVI 文件指针。
- ppavi：用于返回与 AVI 文件关联的流信息。
- fccType：表示获取哪种流的信息。为 streamtypeAUDIO，表示获取音频流，为 streamtypeMIDI，表示获取 MIDI 流，为 streamtypeTEXT，表示获取文本流，为 streamtypeVIDEO，表示获取视频流。
- lParam：表示流类型被访问的计数。

(5) AVIStreamStart 函数。该函数用于获取流的开始帧号。语法如下：

STDAPI\_(LONG) AVIStreamStart(PAVISTREAM pavi);

参数说明：

- pavi：表示 AVI 文件流。

返回值：如果函数执行成功，返回值为流的开始帧号，否则为-1。

(6) AVIStreamLength 函数。该函数用于获得流的长度，也就是流中帧的数量。语法如下：

STDAPI\_(LONG) AVIStreamLength(PAVISTREAM pavi);

参数说明：

- pavi：表示 AVI 文件流。

返回值：如果函数执行成功，返回值为流的长度，否则为-1。

(7) AVIStreamInfo 函数。该函数用于获取流的详细信息。语法如下：

STDAPI AVIStreamInfo(PAVISTREAM pavi, AVISTREAMINFO \* psi, LONG lSize);

参数说明：

- pavi：表示 AVI 文件流。
- psi：表示流信息指针。

● lSize: 表示 AVISTREAMINFO 结构的大小。

(8) AVIStreamReadFormat 函数。该函数用于读取流的帧格式数据。语法如下:

```
STDAPI AVIStreamReadFormat(PAVISTREAM pavi, LONG lPos, LPVOID lpFormat,
LONG * lpcbFormat);
```

参数说明:

- pavi: 表示 AVI 文件流。
- lPos: 表示获取格式数据的流的位置。
- lpFormat: 用于返回获得的格式数据。
- lpcbFormat: 表示格式数据的大小。

(9) AVIStreamGetFrameOpen 函数。该函数用于准备为视频流压缩视频帧。语法如下:

```
STDAPI (PGETFRAME) AVIStreamGetFrameOpen(PAVISTREAM pavi,
LPBITMAPINFOHEADER lpbiWanted);
```

参数说明:

- pavi: 表示 AVI 文件流。
- lpbiWanted: 表示视频格式的地址, 如果为 NULL, 将采用默认的格式。
- 返回值: 函数返回 GETFRAME 对象指针。

(10) AVIFileCreateStream 函数。该函数用于创建一个视频流。语法如下:

```
STDAPI AVIFileCreateStream(PAVIFILE pfile, PAVISTREAM * ppavi,
AVISTREAMINFO * psi);
```

参数说明:

- pfile: 表示 AVI 文件指针。
- ppavi: 表示创建的 AVI 文件流。
- psi: 表示 AVI 文件流信息, 函数将根据该信息创建文件流。

(11) AVIStreamSetFormat 函数。该函数用于设置流格式。语法如下:

```
STDAPI AVIStreamSetFormat(PAVISTREAM pavi, LONG lPos, LPVOID lpFormat,
LONG cbFormat);
```

参数说明:

- pavi: 表示 AVI 文件流。
- lPos: 表示设置格式数据的流的位置。
- lpFormat: 表示设置的格式数据。
- lpcbFormat: 表示格式数据的大小。

(12) AVIStreamWrite 函数。该函数用于向文件流中写入数据。语法如下:

```
STDAPI AVIStreamWrite(PAVISTREAM pavi, LONG lStart, LONG lSamples, LPVOID lpBuffer,
LONG cbBuffer, DWORD dwFlags, LONG * plSampWritten, LONG * plBytesWritten);
```

参数说明:

- pavi: 表示欲写入数据的 AVI 文件流。
- lStart: 表示写入 AVI 文件流的起始位置。
- lSamples: 表示写入的帧数。
- lpBuffer: 表示写入的数据。
- cbBuffer: 表示写入数据的大小。
- dwFlags: 表示与数据相关的标识。
- plSampWritten: 表示实际写入的帧数, 可以为 NULL。
- plBytesWritten: 表示实际写入的字节数, 可以为 NULL。

(13) AVIStreamGetFrameClose 函数。该函数用于关闭打开的视频帧。语法如下:

```
STDAPI AVIStreamGetFrameClose(PGETFRAME pget);
```

参数说明:

- pget: 表示之前调用 AVIStreamGetFrameOpen 函数获得视频帧。

(14) AVIStreamRelease 函数。该函数用于释放 AVI 文件流。语法如下:

STDAPL\_(LONG) AVIStreamRelease(PAVISTREAM pavi);

参数说明:

- pavi: 表示欲释放的 AVI 文件流。

(15) AVIFileRelease 函数。该函数用于释放 AVI 文件指针。语法如下:

STDAPL\_(ULONG) AVIFileRelease(PAVIFILE pfile);

参数说明:

- pfile: 表示欲释放的文件指针。

## 2. ICM 函数

(1) ICMOpen 函数。该函数用于打开视频压缩管理器。语法如下:

HIC ICMOpen( DWORD fccType, DWORD fccHandler, UINT wMode);

参数说明:

- fccType: 4 个字符的代码, 表示压缩器的类型。
- fccHandler: 表示编码、解码器的句柄。
- wMode: 表示压缩或解压缩标记。
- 返回值: 函数返回压缩或解压缩器的句柄。

(2) ICCompressGetFormat 函数。该函数用于获得压缩数据的输出格式。语法如下:

DWORD ICCompressGetFormat(hic, lpbiInput, lpbiOutput);

参数说明:

- hic: 表示压缩器句柄。
- lpbiInput: 表示输入格式的地址, 类型为 BITMAPINFO 结构指针。
- lpbiOutput: 表示输出格式的地址, 类型为 BITMAPINFO 结构指针。

(3) ICCompressQuery 函数。该函数用于确定压缩器是否支持输入格式或者输入格式到输出格式的转换。语法如下:

DWORD ICCompressQuery( hic, lpbiInput, lpbiOutput);

参数说明:

- hic: 表示压缩器句柄。
- lpbiInput: 表示输入格式的地址, 类型为 BITMAPINFO 结构指针。
- lpbiOutput: 表示输出格式的地址, 类型为 BITMAPINFO 结构指针。
- 返回值: 如果函数返回 ICERR\_OK, 表示压缩器支持从输入格式到输出格式的转换, 否则不支持。

(4) ICCompressBegin 函数。该函数用于通知视频驱动程序准备压缩数据。语法如下:

DWORD ICCompressBegin( hic, lpbiInput, lpbiOutput);

参数说明:

- hic: 表示压缩器句柄。
- lpbiInput: 表示输入格式的地址, 类型为 BITMAPINFO 结构指针。
- lpbiOutput: 表示输出格式的地址, 类型为 BITMAPINFO 结构指针。

(5) ICCompress 函数。该函数用于对视频数据进行压缩。语法如下:

DWORD ICCompress(HIC hic, DWORD dwFlags, LPBITMAPINFOHEADER lpbiOutput, LPVOID lpData, LPBITMAPINFOHEADER lpbiInput, LPVOID lpBits, LPDWORD lpckid, LPDWORD lpdwFlags, LONG lFrameNum, DWORD dwFrameSize, DWORD dwQuality, LPBITMAPINFOHEADER lpbiPrev, LPVOID lpPrev);

参数说明:

- hic: 表示压缩器句柄。
- dwFlags: 表示压缩标识。
- lpbiOutput: 表示视频输出格式。
- lpData: 表示压缩后的输出数据。



- lpbiInput: 表示视频输出格式。
  - lpBits: 表示压缩的视频输入数据。
  - lpckid: 保留, 暂未使用。
  - lpdwFlags: 表示返回标记。
  - lFrameNum: 表示压缩帧数。
  - dwFrameSize: 表示帧大小, 可以为 0。
  - dwQuality: 表示压缩品质。
  - lpbiPrev: 表示之前帧的格式, 可以为 NULL。
  - lpPrev: 表示之前帧数据, 可以为 NULL。
  - 返回值: 如果函数执行成功, 返回值为 ICERR\_OK, 则表示函数执行失败。
- (6) ICCompressEnd 函数。该函数用于通知视频压缩驱动程序结束视频压缩。语法如下:  
DWORD ICCompressEnd(hic );

参数说明:

- hic: 表示压缩器句柄。

(7) ICClose 函数。该函数用于关闭压缩器。语法如下:

LRESULT ICClose(HIC hic );

参数说明:

- hic: 表示压缩器句柄。

## 实现过程

- (1) 创建一个基于对话框的应用程序, 在对话框中添加按钮、文本编辑框、静态文本等控件。
- (2) 在工程中引用 vfw.h 头文件, 并导入 vfw32.lib 库文件。

```
#include "vfw.h"
```

```
#pragma comment(lib, "vfw32")
```

- (3) 向对话框中添加 CompressAvi 方法, 压缩 AVI 文件。

```
void CAviCompressDlg::CompressAvi(LPCSTR lpstrOldFile, LPCSTR lpstrNewFile)
```

```
{
 HIC hic;
 ICINFO icInfo;
 icInfo.fccType = 1667524982; //编码解码器类型值
 icInfo.fccHandler = 859066445; //编码解码器句柄值

 BITMAPINFOHEADER InHeader;
 BITMAPINFOHEADER OutHeader;

 //初始化AVI文件
 AVIFileInit();

 PAVIFILE pOldFile, pNewFile;

 HRESULT hRet;
 //打开源文件
 hRet = AVIFileOpen(&pOldFile, lpstrOldFile, OF_READ, NULL);

 if (hRet != 0)
 {
 MessageBox("打开源文件错误", "提示");
 return;
 }
 //打开新文件
 hRet = AVIFileOpen(&pNewFile, lpstrNewFile, OF_WRITE | OF_CREATE, NULL);
 if (hRet != 0)
 {
 MessageBox("打开目标文件错误", "提示");
 return;
 }

 //定义文件流
 PAVISTREAM pOldStream, pNewStream;
```

```

AVIFileGetStream(pOldFile, &pOldStream, streamtypeVIDEO, 0);
//获取流的起始帧
long StartFrame=AVIStreamStart(pOldStream);
//获取流的帧长度
long FrameNum= AVIStreamLength(pOldStream);

AVIStreamInfo OldStreamInfo,NewStreamInfo;
//获取流信息
AVIStreamInfo(pOldStream,&OldStreamInfo,sizeof(AVIStreamInfo));

long size = sizeof(BITMAPINFOHEADER);

AVIStreamReadFormat(pOldStream,StartFrame,&InHeader,&size);
OutHeader=InHeader;

PGETFRAME pFrame;
//在流中打开帧
pFrame=AVIStreamGetFrameOpen(pOldStream, NULL);

void * lpOutData;
lpOutData=VirtualAlloc(NULL,OutHeader.biSizeImage,MEM_COMMIT,
 PAGE_READWRITE);

//打开压缩管理器
hIC=ICOpen(icInfo.fccType,icInfo.fccHandler,ICMODE_COMPRESS);
ICCompressGetFormat(hIC,&InHeader,&OutHeader);

memset(&NewStreamInfo,0,sizeof(NewStreamInfo));
NewStreamInfo.fccType=streamtypeVIDEO;
NewStreamInfo.fccHandler=mmioFOURCC('M', 'S', 'V', 'C');
NewStreamInfo.dwScale=1;
NewStreamInfo.dwRate=25;
NewStreamInfo.dwSuggestedBufferSize=OutHeader.biSizeImage;
SetRect(&NewStreamInfo.rcFrame,0,0,OutHeader.biWidth,OutHeader.biHeight);

if(ICCompressQuery(hIC,&InHeader,&OutHeader)==ICERR_OK)
{
 ICCompressBegin(hIC,(BITMAPINFO*)&InHeader,(BITMAPINFO*)&OutHeader);
}

AVIFileCreateStream(pNewFile,&pNewStream,&NewStreamInfo);
AVIStreamSetFormat(pNewStream,0,&OutHeader,sizeof(OutHeader));

for (int index=StartFrame; index<FrameNum; index++)
{
 long num = sizeof(InHeader);
 AVIStreamReadFormat(pOldStream,index,&OutHeader,&num);

 BYTE* pDIB = (BYTE*) AVIStreamGetFrame(pFrame, index);
 BYTE* pData=pDIB+sizeof(BITMAPINFOHEADER);

 DWORD dwCkID;
 DWORD dwCompFlags;
 DWORD dwQuality=100;
 if(ICCompress(hIC,ICCOMPRESS_KEYFRAME,&OutHeader,lpOutData,
 &InHeader,pData,&dwCkID,&dwCompFlags,index,0,dwQuality,
 NULL,NULL)==ICERR_OK)
 {
 AVIStreamSetFormat(pNewStream,index,&OutHeader,sizeof(OutHeader));
 AVIStreamWrite(pNewStream,index,1,(LPBYTE)lpOutData,
 OutHeader.biSizeImage,AVIIF_KEYFRAME,NULL,NULL);
 }
}
if(hIC!=NULL)
{
 ICCompressEnd(hIC);
 ICclose (hIC);
}
AVIStreamGetFrameClose(pFrame);
AVIStreamRelease(pNewStream);
AVIStreamRelease(pOldStream);

```



```

AVIFileRelease(pOldFile);
AVIFileRelease(pNewFile);
VirtualFree(lpOutData, OutHeader.biSizeImage, MEM_DECOMMIT);
AVIFileExit();
}

```

(4) 处理 “...” 按钮的单击事件，从磁盘中选择 AVI 文件。

```

void CAviCompressDlg::OnSrcBrown()
{
 CFileDialog fDlg(TRUE, "", "", OFN_HIDEREADONLY |
 OFN_OVERWRITEPROMPT, "AVI文件*.avi");
 if (fDlg.DoModal() != IDOK)
 {
 CString strFile = fDlg.GetPathName();
 m_SrcFile.SetWindowText(strFile);
 }
}

void CAviCompressDlg::OnDstBrown()
{
 CFileDialog fDlg(FALSE, "avi", "One", OFN_HIDEREADONLY |
 OFN_OVERWRITEPROMPT, "AVI文件*.avi");
 if (fDlg.DoModal() != IDOK)
 {
 CString strFile = fDlg.GetPathName();
 DeleteFile(strFile);
 m_DstFile.SetWindowText(strFile);
 }
}

```

(5) 处理 “压缩” 按钮的单击事件，开始压缩文件。

```

//对AVI文件进行再次压缩
void CAviCompressDlg::OnCompress()
{
 CString strSrc, strDst;
 m_SrcFile.GetWindowText(strSrc);
 m_DstFile.GetWindowText(strDst);
 if (strSrc.IsEmpty() || strDst.IsEmpty())
 {
 MessageBox("源文件或目标文件不能为空!", "提示");
 return;
 }
 CompressAvi(strSrc, strDst);
 MessageBox("压缩完成!");
}

```

### 举一反三

根据本实例，读者可以：

- 开发数码相机应用程序。

## 实例 172

### 使用 Direct Show 设计媒体播放器

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\04\172

### 实例说明

在互联网上有许多优秀的媒体播放器软件，功能也各具千秋。例如，金山影霸、影音风暴和 Windows 的 MediaPlayer 等。本节中笔者利用 DirectShow 设计一个媒体播放器，突出的特色功能就是实现影片抓图，实现图像的亮度、饱和度、对比度设置，效果如图 4-30 所示。

### 技术要点

对于 Visual C++ 的初学者来说，应用 DirectShow 进行多媒体应用程序开发有一定的难度，需要理解 DirectShow 的开发思路，需要具有一定的 Com 知识。下面介绍如何在 Visual C++ 中应用 DirectShow 进行程序开发。

- 如何使用 Direct Show 开发包

DirectX 是微软公司推出的一套基于 Windows 平台的图像、声音、输入输出和网络游戏的编程接口。DirectX 被定义为与设备无关性,即 DirectX 可以使用与设备无关的方法提供设备相关的高性能。DirectX 是一个大家族,其成员主要包括 Direct Input、Direct Play、Direct Setup、Direct Music、Direct Sound、DirectX Media Objects、DirectX Graphics 和 Direct Show 等。其中,Direct Show 主要是为 Windows 平台处理媒体文件播放、音视频采集提供了完整的解决方案。

在使用 Direct Show 之前,需要安装 DirectX 开发包和 Direct Show 开发包。用户可以到微软公司的官方网站上下载。这里需要说明的是,在 DirectX8.1 SDK 版本中包含有 Direct Show SDK,在 DirectX9.0C SDK 的第一个版本 DirectX SDK Summer 2004 中也包含有 Direct Show SDK,之后,在 DirectX SDK 中没有包含 Direct Show SDK,Direct Show SDK 以 Extras 的形式单独发布。

本程序采用的 DirectX SDK 为 Microsoft DirectX SDK (April 2006), Direct Show SDK 为 DirectX SDK Extras February 2005。

在安装完 DirectX SDK 和 Direct Show SDK 之后,为了能够在程序中使用 Direct Show,首先需要将 Visual C++ 2005 目录下的“BaseTsd.h”文件复制到 Direct Show SDK 安装目录的 Include 目录下。否则,在 Visual C++ 6.0 中使用 Direct Show 将无法通过编译。然后在 Visual C++6.0 开发环境中设置文件查找路径。具体步骤如下。

(1) 单击 Tools/Options 菜单项打开选项窗口,选择 Directories 选项卡,如图 4.31 所示。

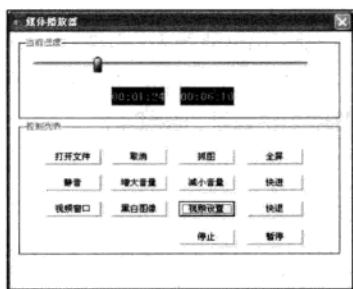


图 4.30 使用 Direct Show 设计媒体播放器

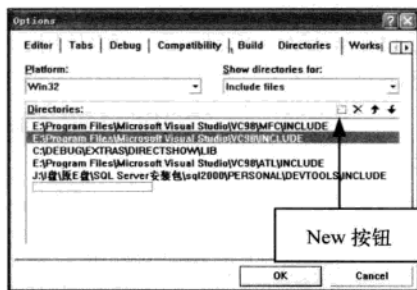


图 4.31 选项窗口

(2) 单击“New”按钮创建一个列表,设置目录为 DirectX SDK 安装目录下的 Include 目录,如图 4.32 所示。

(3) 单击“New”按钮创建一个列表项,分别设置目录为 Direct Show SDK 安装目录下的 Include 目录,如图 4.33 所示。

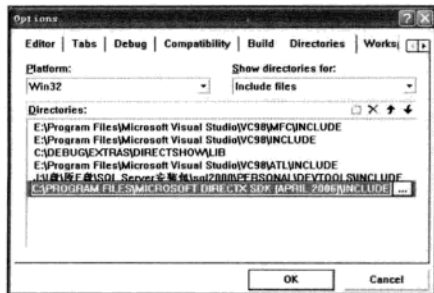


图 4.32 设置 DirectX SDK 目录

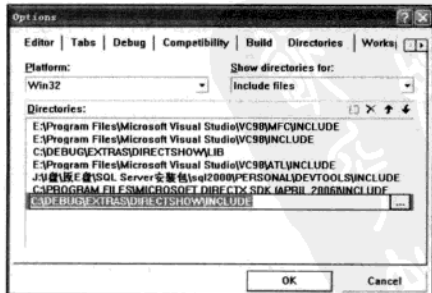


图 4.33 设置 Direct Show SDK 目录

(4) 在 Show directories for 组合框中选择 Library files 选项,单击“New”按钮新建一个列表项,设置目录为 Direct Show SDK 安装目录下的\LIB\X86 目录,用于查找库文件 Strmiids.lib 和 quartz.lib,如图 4.34 所示。



(5) 调整 Direct Show 列表中的选项顺序, 如图 4.35 所示。

在完成了上面的配置之后, 还需要引用 dshow.h 头文件, 链接 Strmiids 和 quartz 库文件就可以在程序中使用 Direct Show SDK 了。

```
#include "dshow.h"
#pragma comment (lib,"Strmiids")
#pragma comment (lib,"quartz")
```

#### ● 使用 Direct Show 开发程序的方法

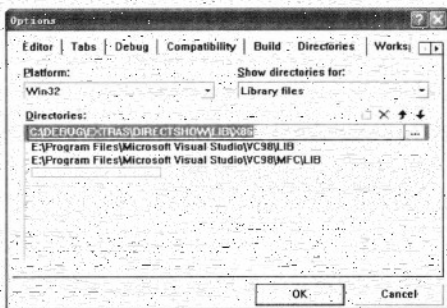


图 4.34 设置库文件路径

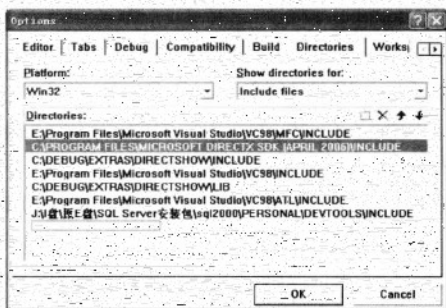


图 4.35 调整列表项顺序

Direct Show 提供了一个很有用的工具——graphedt, 在 Direct Show 安装目录的 Utilities 目录下。在使用 Direct Show 开发应用程序前, 可以使用 graphedt 预先设置出过滤图表。以播放一个 DAT 文件为例, 使用 graphedt 打开 DAT 文件, graphedt 将根据系统信息生成过滤图像, 如图 4.36 所示。

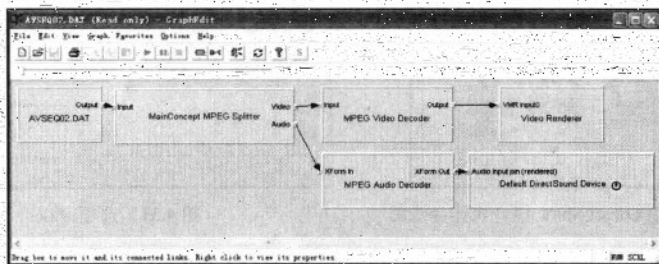


图 4.36 过滤图表

如果需要使用 Direct Show 实现播放 DAT 文件的功能, 只要按照图表中显示的过滤器——创建过滤器, 并按顺序连接过滤器就可以了。当然, 另一种更简单的方法就是使用过滤图表对象的 RenderFile 方法, 该方法将根据文件名自动生成过滤图表。下面给出播放媒体文件的关键代码。

```
IGraphBuilder *pGraph; //定义过滤图表接口
ICaptureGraphBuilder2 *pBuilder; //定义图表构建接口对象
IMediaControl *pMediaControl; //定义媒体控制接口
CoCreateInstance(CLSID_CaptureGraphBuilder2, 0, CLSCTX_INPROC_SERVER,
IID_ICaptureGraphBuilder2, (void**)&pBuilder); //创建图表构建的接口对象
CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
IID_IFilterGraph2, (void**)&pGraph); //创建过滤图表接口对象
pBuilder->SetFiltergraph(pGraph); //设置过滤图表
pGraph->RenderFile(strFile, AllocSysString(), NULL); //通过媒体文件构建默认的过滤图表
pGraph->QueryInterface(IID_IMediaControl, (void**)&pMediaControl); //获取媒体控制对象
pMediaControl->Run(); //开始播放文件
```

#### ● 确定媒体文件播放完成

在设计媒体播放器时, 需要在媒体文件播放完成后得到通知。原因是在设计文件播放列表时, 需要在媒体文件播放完成后播放下一个文件。在 Direct Show 中, 可以通过媒体控制接口和一个单独的线程来实现。具体步骤如下。

(1) 创建一个媒体控制接口对象。

```
IMediaEventEx *pEvent; //定义媒体事件接口对象
pGraph->QueryInterface(IID_IMediaEventEx, (void **)&pEvent); //获取媒体事件接口对象
```

(2) 创建一个线程, 执行线程函数, 在线程函数中判断媒体文件是否播放完成, 如果没有播放完成, 发送 CM\_POSCHANGE 自定义消息, 如果播放完成, 发送 CM\_COMPLETE 自定义消息。

```
DWORD WINAPI ThreadProc(LPVOID lpParameter)
{
 CDirectShowEventDlg* pWnd = (CDirectShowEventDlg*)lpParameter; //获取主窗口指针
 HANDLE hEvent; //定义事件句柄
 pWnd->pEvent->GetEventHandle((OAEVENT*) &hEvent); //获取事件句柄
 long code, p1, p2; //定义事件代码参数
 BOOL done = FALSE;
 while (!done) //开始执行循环
 {
 pWnd->SendMessage(CM_POSCHANGE); //发送CM_POSCHANGE消息
 if (WaitForSingleObject(hEvent, 80) == WAIT_OBJECT_0) //Direct Show是否有事件产生
 {
 //获取事件代码
 while (SUCCEEDED(pWnd->pEvent->GetEvent(&code, &p1, &p2, 0)))
 {
 pWnd->pEvent->FreeEventParams(code, p1, p2); //释放事件参数
 if (code == EC_COMPLETE) //是否为播放完成
 {
 pWnd->m_Completed = TRUE; //设置播放完成状态
 pWnd->SendMessage(CM_COMPLETE); //发送CM_COMPLETE消息
 done = true; //退出循环, 结束线程
 }
 }
 }
 }
 return 0;
}
```

#### ● 使用 Direct Show 进行音量和播放进度的控制

对于媒体播放软件来说, 实现音量调节和播放进度的控制是不可获取的功能。在 Direct Show 中可以使用 IBasicAudio 接口来实现音量的控制。该接口提供了 get\_Volume 和 put\_Volume 方法用于获取和设置音量。如果想设置为静音, 可以将音量设置为“-10000L”。下面的代码演示了音量的控制。

```
IBasicAudio* pAudio = NULL; //定义IBasicAudio接口指针
if (pGraph != NULL)
{
 pGraph->QueryInterface(IID_IBasicAudio, (void **)&pAudio); //获取IBasicAudio接口对象
 if (pAudio != NULL) //如果有音频数据
 {
 pAudio->get_Volume(&m_IVolumn); //获取当前的音量
 if (m_IVolumn < 0) //当前不是最大音量
 {
 m_IVolumn += 200; //增加音量
 pAudio->put_Volume(m_IVolumn); //设置音量
 }
 else
 {
 m_IVolumn = 0; //当前音量为最大音量
 }
 }
}
```

如果想要控制播放的进度, 可以使用 IMediaPosition 接口来实现。该接口提供有 get\_StopTime 方法获取文件播放结束时的停止事件, 提供有 get\_CurrentPosition 方法获取当前的播放时间, 提供有 put\_CurrentPosition 方法设置播放时间, 也就是设置播放进度。有了这些方法, 实现播放进度的控制就易如反掌了。相关代码如下:

```
IMediaPosition* pPosition = NULL; //定义IMediaPosition接口指针
if (pGraph != NULL)
{
 pGraph->QueryInterface(IID_IMediaPosition, (void **)&pPosition); //获取IMediaPosition接口对象
 if (pPosition != NULL)
 {
 REFTIME curTime, endTime;
 pPosition->get_StopTime(&endTime); //获取播放的停止时间
 pPosition->get_CurrentPosition(&curTime); //获取当前的播放时间
 curTime += 5; //设置快进
 }
}
```

```

if (curTime <= endTime)
{
 pPosition->put_CurrentPosition(curTime); //设置当前播放的时间
}
else
{
 pPosition->put_CurrentPosition(endTime); //设置当前播放时间为停止时间
}
}
}

```

### ● 使用 Direct Show 实现亮度、饱和度和对比度调节

在媒体播放器模块中，笔者实现了对视频图像的亮度、饱和度和对比度的设置。主要方法是使用 VideoMixingRenderer9 过滤器支持的 IVMRMixerControl9 接口实现的，该接口提供有 SetProcAmpControl 方法用于设置图像的亮度、饱和度和对比度等信息。主要代码如下：

```

void CDirectShowEventDlg::SetViewInfo(int nFlag, float fValue)
{
 IVMRMixerControl9 * pControl = NULL; //定义IVMRMixerControl9接口指针
 if (pRender != NULL)
 {
 //获取IVMRMixerControl9接口对象
 pRender->QueryInterface(IID_IVMRMixerControl9, (void**)&pControl);
 if (pControl != NULL)
 {
 VMR9ProcAmpControl vmrParam; //定义参数
 memset(&vmrParam, 0, sizeof(VMR9ProcAmpControl)); //初始化参数
 vmrParam.dwSize = sizeof(VMR9ProcAmpControl); //设置参数大小
 vmrParam.dwFlags = nFlag; //设置标记
 vmrParam.Brightness = fValue; //设置亮度值
 vmrParam.Contrast = fValue; //设置对比度值
 vmrParam.Hue = fValue; //设置色调值
 vmrParam.Saturation = fValue; //设置饱和度值
 pControl->SetProcAmpControl(0, &vmrParam); //设置视频图像颜色信息
 }
 }
}

```

## 实现过程

- (1) 创建一个基于对话框的工程，工程名称为 MediaPlayer。
- (2) 创建一个对话框类，类名为“CDirectShowEventDlg”。
- (3) 向对话框中添加按钮、静态文本和滑块控件，设置主要控件属性，如表 4.7 所示。

表 4.7 媒体播放器主窗口控件属性设置

| 控 件 ID          | 控 件 属 性                        | 关 联 变 量                       |
|-----------------|--------------------------------|-------------------------------|
| IDC_CTLLIST     | Caption: 控制列表<br>Border: FALSE | CCustomGroup; m_CtlList       |
| IDC_GRAY        | Caption: 黑白图像                  | CButton; m_GrayBtn            |
| IDC_PROCESSCTRL | 默认                             | CCustomSlider; m_ProgressCtrl |
| IDC_CURPOS      | Caption: 空<br>Border: FALSE    | CNumLabel; m_CurPos           |

- (4) 引用 Direct Show 相关头文件和库文件。

```

#include "dshow.h"
#include "D3d9.h"
#include "vmr9.h"
#include "Objbase.h"
#pragma comment (lib, "Strmiids")
#pragma comment (lib, "quartz")

```

- (5) 在应用程序初始化时初始化 Com 库，因为 Direct Show 操作是基于 Com 技术实现的。

其实现代码如下：

```

CoInitialize(NULL); //初始化Com库

```

- (6) 向对话框中添加 FindPin 方法，用于查找某一过滤器的输入、输出引脚。其实现代码如下：

```

//查找引脚
IPin* CDirectShowEventDlg::FindPin(IBaseFilter *pFilter, PIN_DIRECTION dir)

```

```

{
 IEnumPins* pEnumPins;
 IPin* pOutpin;
 PIN_DIRECTION pDir;
 pFilter->EnumPins(&pEnumPins);
 while (pEnumPins->Next(1,&pOutpin,NULL)==S_OK)
 {
 pOutpin->QueryDirection(&pDir);
 if (pDir==dir)
 {
 return pOutpin;
 }
 }
}
}

```

//定义IEnumPins接口指针  
//定义IPin接口指针  
//定义引脚方向  
//列举过滤器引脚  
  
//获取引脚方向(输入或输出引脚)  
//判断引脚方向

(7) 向对话框中添加 Done 方法, 当用户播放一个媒体文件时, 将创建一个线程, 用于检测媒体文件是否播放完成, 如果播放完成将向主窗口发送自定义消息 CM\_COMPLETE, 该消息关联 Done 方法。Done 方法的作用是在媒体文件播放完成之后, 终止线程, 隐藏 Direct Show 的视频显示窗口, 释放媒体控制接口对象指针, 释放过滤图标, 恢复对话框的成员变量为初始状态, 修改视频显示窗口, 使得在媒体文件播放完成后不可以调整窗口大小。其实现代码如下:

```

void CDirectShowEventDlg::Done(WPARAM wParam, LPARAM lParam)
{
 if (m_hThread)
 {
 TerminateThread(m_hThread,0);
 m_hThread = NULL;
 }
 if (pMediaControl != NULL)
 {
 pMediaControl->Stop();
 m_bStop = TRUE;
 if (pViewWnd != NULL)
 {
 pViewWnd->put_Visible(FALSE);
 }
 }
 if (pMediaControl)
 {
 pMediaControl->Release();
 pMediaControl = NULL;
 }
 if (pGraph)
 {
 pGraph->Release();
 pGraph = NULL;
 }
 if (pEvent)
 {
 pEvent->Release();
 pEvent = NULL;
 }
 m_bFullScreen = FALSE;
 pViewWnd = NULL;
 pBaseVideo = NULL;
 pBase = NULL;
 m_bViewPlay = FALSE;
 m_lVolume = 0;
 m_bMute = FALSE;
 m_bSpeed = FALSE;
 m_bBack = FALSE;
 m_bGrayImage = FALSE;
 m_fSaturation = m_fBright = m_fContrast = m_fHue = 1;
 pRender = NULL;
 m_bStop = m_bPause = FALSE;
 m_hThread = NULL;
 m_Stop.SetWindowText("停止");
 m_GrayBtn.SetWindowText("黑白图像");
 m_Pause.SetWindowText("暂停");
 m_Progress.SetText("00:00:00");
 m_CurPos.SetText("00:00:00");
 m_ProgressCtrl.SetPos(0);
 //播放完成
 m_Previewed = FALSE;
 m_Completed = TRUE;
}

```

//判断线程是否运行  
//终止线程  
//是否正在播放  
//停止播放  
//是否显示视频图像  
//隐藏视频窗口  
//判断媒体控制接口指针是否为空  
//释放媒体控制接口指针  
//判断过滤图表接口指针是否为空  
//释放过滤图表接口指针  
//释放媒体控制对象  
//将成员变量恢复为原来的状态  
  
//设置停止按钮的文本



```
m_DisplayWnd.ModifyStyle(WS_SIZEBOX,0); //修改视频显示窗口的风格
m_DisplayWnd.m_Panel.ModifyStyle(SS_BLACKRECT,SS_BITMAP);
m_DisplayWnd.m_Panel.SetBitmap(m_DisplayWnd.bmp); //设置默认的位图
m_DisplayWnd.SetWindowPos(NULL,0,0,m-OriginRC.Width(),
m-OriginRC.Height(),SWP_NOMOVE); //恢复视频窗口大小
```

(8) 向对话框中添加 OnPosChange 方法, 用于时时显示当前的播放时间, 当播放一个媒体文件时, 将创建一个线程判断媒体文件是否播放完成, 如果没有播放完成, 则发送自定义消息 CM\_POSCHANGE, 该消息关联 OnPosChange 方法。其实现代码如下:

```
void CDirectShowEventDlg::OnPosChange()
{
 if (pGraph != NULL)
 {
 IMediaPosition* pPosition = NULL; //定义IMediaPosition接口指针
 //获取IMediaPosition接口对象
 pGraph->QueryInterface(IID_IMediaPosition,(void**)&pPosition);
 if (pPosition != NULL)
 {
 REFTIME endTime;
 pPosition->get_CurrentPosition(&endTime); //获取当前的播放事件
 m_ProgressCtrl.SetPos(endTime); //设置进度条的显示位置
 int nHour = endTime / 3600; //将秒转换为小时:分:秒的形式
 int nMinute = (endTime - nHour*3600)/60;
 int nSecond = (int)endTime % 60;
 CString csTime,csSpace;
 csSpace = "";
 if (nHour<10)
 csSpace += "0%d:";
 else
 csSpace += "%d:";
 if (nMinute<10)
 csSpace += "0%d:";
 else
 csSpace += "%d:";
 if (nSecond<10)
 csSpace += "0%d";
 else
 csSpace += "%d";
 csTime.Format(csSpace,nHour,nMinute,nSecond); //格式化字符串
 m_CurPos.SetText(csTime); //显示播放事件
 }
 }
}
```

(9) 向对话框中添加 PlayFile 方法, 播放指定的媒体文件。该方法首先判断当前是否正在播放媒体文件, 如果是则调用 Done 方法结束媒体文件的播放, 然后构建适当的过滤图表, 在构建完过滤图表之后, 将创建一个线程, 该线程的作用是修改过滤图表, 默认情况下, Direct Show 使用 Video Renderer 过滤器来显示视频图像, 该过滤器不能实现字幕叠加功能, 不能够调节视频图像的亮度、饱和度和对比度, 我们需要将该过滤器替换为 VideoMixingRenderer9 过滤器, 线程函数的作用就是使用 VideoMixingRenderer9 过滤器替换 Video Renderer 过滤器。接着在 PlayFile 方法中将判断媒体文件是否包含视频显示, 如果是则显示视频窗口, 最后获取媒体文件的长度, 创建一个单独的线程来检测媒体文件是否播放完成, 其实现代码如下:

```
void CDirectShowEventDlg::PlayFile(LPCTSTR lpFileName)
{
 if (pGraph != NULL) //之前已经播放文件或者正在播放文件
 {
 Done(0,0); //停止播放
 }
 m_bStop = FALSE; //初始化变量
 pBuilder = NULL;
 pGraph = NULL;
 pMediaControl = NULL;
 m_FileName = lpFileName;
 CoCreateInstance(CLSID_CaptureGraphBuilder2,0,CLSCTX_INPROC_SERVER,
 IID_ICaptureGraphBuilder2,(void**)&pBuilder); //创建图表构建接口对象
 CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
 IID_IFilterGraph2, (void **)&pGraph); //创建过滤图表对象
 pBuilder->SetFiltergraph(pGraph); //构造默认的过滤图表
 m_bInvalidFile = FALSE;
 pRender = NULL;
```

```

CoCreateInstance(CLSID_VideoMixingRenderer9, NULL, CLSCTX_ALL,
 IID_IBaseFilter, (void **)&pRender); //创建VideoMixingRenderer9过滤器
m_ThreadStop = FALSE;
//创建一个单独的线程实现视频显示过滤器的替换
HANDLE hHandle = CreateThread(NULL, 0, PlayVideoFile, (void*)this, 0, NULL);
while (!m_ThreadStop) //等待线程函数执行完成
{
 MSG msg;
 ::SendMessage(&msg, NULL, 0, WM_USER); //在循环过程中响应界面操作
 ::TranslateMessage(&msg);
 ::DispatchMessage(&msg);
}
if (m_bInvalidFile) //文件非法
{
 Done(0, 0); //停止播放
 return;
}
pViewWnd = NULL;
//获取预览窗口
pGraph->QueryInterface(IID_IVideoWindow, (void **)&pViewWnd);
if (pViewWnd)
{
 //设置预览窗口的拥有者
 pViewWnd->put_Owner((long)m_DisplayWnd.m_Panel.m_hWnd);
 pViewWnd->put_Left(0);
 pViewWnd->put_Top(0);
 //获取预览窗口风格
 long style;
 pViewWnd->get_WindowStyle(&style);
 style = style & ~WS_CAPTION;
 style = style & ~WS_DLGFRAME;
 style = style & WS_CHILD;
 pViewWnd->put_WindowStyle(style);
 //设置预览窗口宽度和高度
 CRect rc;
 m_DisplayWnd.m_Panel.GetClientRect(rc);
 pViewWnd->put_Height(rc.Height());
 pViewWnd->put_Width(rc.Width());
 pViewWnd->put_MessageDrain((OAHWND)m_DisplayWnd.m_Panel.m_hWnd);
}
if (m_bViewPlay)
{
 m_DisplayWnd.ShowWindow(SW_SHOW); //显示视频显示窗口
 m_DisplayWnd.ModifyStyle(0, WS_SIZEBOX); //设置视频显示窗口风格
}
else
{
 m_DisplayWnd.ShowWindow(SW_HIDE); //隐藏视频显示窗口
 m_DisplayWnd.ModifyStyle(WS_SIZEBOX, 0); //设置视频显示窗口风格
}
m_hThread = NULL;
//获取媒体控制接口对象
pGraph->QueryInterface(IID_IMediaControl, (void **)&pMediaControl);
pMediaControl->Run(); //开始播放文件
IMediaPosition* pPosition = NULL; //定义IMediaPosition接口指针
//获取IMediaPosition接口对象
pGraph->QueryInterface(IID_IMediaPosition, (void **)&pPosition);
if (pPosition != NULL)
{
 REFTIME curTime, endTime;
 pPosition->get_StopTime(&endTime); //获取文件播放的停止事件
 pPosition->get_CurrentPosition(&curTime); //获取文件播放的当前事件
 m_ProgressCtrl.SetRange(curTime, endTime); //设置进度条的范围
 //将秒转换为小时:分:秒的形式
 int nHour = endTime / 3600; //获取小时
 int nMinute = (endTime - nHour*3600)/60; //获取分钟
 int nSecond = (int)endTime % 60; //获取秒
 CString csTime, csSpace;
 csSpace = "";
 if (nHour < 10)
 csSpace += "0%d:";
 else
 csSpace += "%d:";
 if (nMinute < 10)
 csSpace += "0%d:";
 else
 csSpace += "%d:";
 if (nSecond < 10)
 csSpace += "0%d:";
 else
 csSpace += "%d:";
}

```

```

 csSpace += "0%d";
 else
 csSpace += "%d";
 csTime.Format(csSpace,nHour,nMinute,nSecond); //格式化字符串
 m_Progress.SetText(csTime); //设置显示时间
}
pEvent = NULL;
//获取IID_IMediaEventEx接口对象
pGraph->QueryInterface(IID_IMediaEventEx, (void **)&pEvent);
m_Completed = FALSE;
DWORD threadID;
//开始一个线程, 检测文件是否播放完成
m_hThread = CreateThread(NULL,0,ThreadProc,(void*)this,0,&threadID);
m_Previewed = TRUE;
}

```

(10) 添加 PlayVideoFile 全局函数, 该函数作为一个线程函数, 当构建完媒体文件的过滤图表之后, 将创建一个线程执行线程函数 PlayVideoFile, 用于使用 VideoMixingRenderer9 过滤器替换 Video-Renderer 过滤器。其实现代码如下:

```

DWORD WINAPI PlayVideoFile(LPVOID lpParameter)
{
 //获取主窗口指针
 CDirectShowEventDlg* pWnd = (CDirectShowEventDlg*)lpParameter;
 CString strFile= pWnd->m_FileName; //获取播放的文件名
 //构建过滤图
 HRESULT hRet = pWnd->pGraph->RenderFile(strFile.AllocSysString(),NULL);
 if(hRet!=S_OK)
 {
 pWnd->m_bInvalidFile = TRUE; //非法的文件
 return 1;
 }
 //获取VideoRender filter
 pWnd->pBase == NULL;
 IBaseFilter* pRenderFilter = NULL;
 //获取默认的视频显示过滤器
 pWnd->pGraph->FindFilterByName(L"Video Renderer",(IBaseFilter**)&pRenderFilter);
 if (pRenderFilter!=NULL) //包含视频信息
 {
 pWnd->m_bViewPlay = TRUE;
 IPin* pVideoIn = NULL; //定义引脚接口指针
 //查找输入引脚
 pVideoIn = pWnd->FindPin(pRenderFilter,PINDIR_INPUT);
 if (pVideoIn)
 {
 pVideoIn->Disconnect(); //与视频解码过滤器断开连接
 }
 pWnd->pGraph->RemoveFilter(pRenderFilter); //移除视频显示过滤器
 //添加VideoMixingRenderer9过滤器
 pWnd->pGraph->AddFilter(pWnd->pRender,L"Render");
 //获取视频解码器
 pWnd->pGraph->FindFilterByName(L"MPEG Video Decoder",(IBaseFilter**)&pWnd->pBase);
 if (pWnd->pBase)
 {
 IPin* pOutPin = NULL;
 //获取视频解码的输出引脚
 pOutPin = pWnd->FindPin(pWnd->pBase,PINDIR_OUTPUT);
 if (pOutPin != NULL)
 {
 IPin* pColorIn = NULL;
 IPin* pColorOut = NULL;
 //获取输入和输出引脚
 IPin *pTextIn = NULL;
 IPin *pTextOut = NULL;
 IPin *pRenderIn = NULL;
 pRenderIn = pWnd->FindPin(pWnd->pRender,PINDIR_INPUT);
 HRESULT hRet = 0;
 hRet = pOutPin->Disconnect(); //断开连接
 //连接视频解码过滤器输出引脚与VideoMixingRenderer9过滤器输入引脚
 hRet = pWnd->pGraph->ConnectDirect(pOutPin,pRenderIn,NULL);
 }
 }
 }
 else
 {
 pWnd->m_bViewPlay = FALSE; //没有视频信息
 }
 pWnd->m_ThreadStop = TRUE;
}

```

```
return 0;
```

(11) 添加一个全局函数 ThreadProc, 该函数将作为一个线程函数。当播放一个媒体文件时将创建一个线程来检测媒体文件是否播放完成, 如果没有播放完成, 则向主窗口发送 CM\_POSCHANGE 消息执行 OnPosChange 方法显示当前的播放时间, 否则向主窗口发送 CM\_COMPLETE 消息执行 Done 方法结束媒体文件播放。其实现代码如下:

```
DWORD WINAPI ThreadProc(LPVOID lpParameter)
{
 CDirectShowEventDlg* pWnd = (CDirectShowEventDlg*)lpParameter; //获取主窗口指针
 HANDLE hEvent; //定义事件句柄
 pWnd->pEvent->GetEventHandle((OAEVENT*) &hEvent); //获取事件句柄
 long code, p1, p2; //定义事件代码参数
 BOOL done = FALSE; //开始执行循环
 while (!done)
 {
 pWnd->SendMessage(CM_POSCHANGE); //发送CM_POSCHANGE消息
 if (WaitForSingleObject(hEvent, 80) == WAIT_OBJECT_0) //Direct Show是否有事件产生
 {
 //获取事件代码
 while (SUCCEEDED(pWnd->pEvent->GetEvent(&code, &p1, &p2, 0)))
 {
 pWnd->pEvent->FreeEventParams(code, p1, p2); //释放事件参数
 if (code == EC_COMPLETE) //是否为播放完成
 {
 pWnd->m_Completed = TRUE; //设置播放完成状态
 pWnd->SendMessage(CM_COMPLETE); //发送CM_COMPLETE消息
 done = true; //退出循环, 结束线程
 }
 }
 }
 }
 return 0;
}
```

(12) 处理“打开文件”按钮的单击事件, 利用文件打开对话框选择一个媒体文件, 然后用 PlayFile 方法播放媒体文件。其实现代码如下:

```
void CDirectShowEventDlg::OnSetFile()
{
 CFileDialog fDlg(TRUE, NULL, NULL, OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
 "avi文件*.avi;*.dat;*.mp3;*.wav;*.mpeg|所有文件|*.*|", this); //定义文件打开对话框
 if (fDlg.DoModal() == IDOK)
 {
 m_FileName = fDlg.GetPathName(); //获取文件名
 m_Stop.SendMessage(WM_LBUTTONDOWN, 0, 0); //发送CM_COMPLETE消息
 PlayFile(m_FileName); //开始播放媒体文件
 }
}
```

(13) 处理“抓图”按钮的单击事件, 将抓取当前视频窗口的图像信息, 将其保存到磁盘文件中。主要方法是使用 IbasicVideo 接口的 GetCurrentImage 方法来获得位图的信息后和位图数据。其实现代码如下:

```
void CDirectShowEventDlg::OnSnap()
{
 CFileDialog fDlg(FALSE, "", "Snap.bmp"); //定义文件保存对话框
 if (pGraph != NULL)
 {
 IBasicVideo * pBasicVideo = NULL; //定义IBasicVideo接口指针
 //获取IBasicVideo接口对象
 pGraph->QueryInterface(IID_IBasicVideo, (void **)&pBasicVideo);
 if (pBasicVideo != NULL)
 {
 pMediaControl->Pause(); //暂停播放
 if (fDlg.DoModal() == IDOK)
 {
 CString csSaveName = fDlg.GetPathName(); //获取文件名
 //获取图像大小, 包含位图信息头
 long lBmpSize;
 if (SUCCEEDED(pBasicVideo->GetCurrentImage(&lBmpSize, 0)))
 {
 //定义图像数据缓冲区, 获取图像数据
 BYTE* pData = new BYTE[lBmpSize];
 if (SUCCEEDED(pBasicVideo->GetCurrentImage(&lBmpSize, (long*)pData)))
 {

```



```

 BITMAPFILEHEADER bFile;
 bFile.bfReserved1 = bFile.bfReserved2 = 0;
 bFile.bfSize = sizeof(BITMAPFILEHEADER);
 bFile.bfType = 0x4d42;
 bFile.bfOffBits = sizeof(BITMAPFILEHEADER)+
 sizeof(BITMAPINFOHEADER);
 CFile file; //定义文件对象
 //创建并打开文件
 file.Open(csSaveName,CFile::modeCreate|CFile::modeReadWrite);
 //写入文件数据
 file.Write(&bFile,sizeof(BITMAPFILEHEADER));
 file.WriteHuge(pData,lBmpSize);
 file.Close(); //关闭文件
 }
 delete [] pData; //释放位图数据
 pBasicVideo->Release(); //释放IBasicVideo接口指针
}
pMediaControl->Run(); //继续播放媒体文件
}
}

```

(14) 处理“全屏”按钮的单击事件,调用视频窗口对象的 put\_FullScreenMode 方法设置全屏模式。其实现代码如下:

```

void CDirectShowEventDlg::OnFullScreen()
{
 if (pViewWnd!= NULL && m_bStop==FALSE)
 {
 pViewWnd->put_FullScreenMode(-1); //设置全屏显示
 m_bFullScreen = TRUE;
 }
}

```

(15) 改写对话框的 PreTranslateMessage 方法,在全屏模式下按<Esc>键将取消全屏显示。其实现代码如下:

```

BOOL CDirectShowEventDlg::PreTranslateMessage(MSG* pMsg)
{
 if (pMsg->message == WM_KEYDOWN) //是否为按键消息
 {
 if (pMsg->wParam == VK_ESCAPE) //是否为<Esc>键
 {
 if (pViewWnd!= NULL)
 {
 pMsg->wParam = 0;
 pViewWnd->put_FullScreenMode(0); //取消全屏显示
 m_bFullScreen = FALSE;
 return TRUE;
 }
 }
 }
 return CDialog::PreTranslateMessage(pMsg);
}

```

(16) 向对话框中添加 SetViewInfo 方法,用于设置视频图像的颜色,如亮度、饱和度、对比度的信息。其实现代码如下:

```

void CDirectShowEventDlg::SetViewInfo(int nFlag, float fValue)
{
 IVMRMixerControl9 * pControl = NULL; //定义IVMRMixerControl9接口指针
 if (pRender != NULL)
 {
 //获取IVMRMixerControl9接口对象
 pRender->QueryInterface(IID_IVMRMixerControl9,(void**)&pControl);
 if (pControl != NULL)
 {
 VMR9ProcAmpControl ymrParam; //定义参数
 memset(&ymrParam,0,sizeof(VMR9ProcAmpControl)); //初始化参数
 ymrParam.dwSize = sizeof(VMR9ProcAmpControl); //设置参数大小
 ymrParam.dwFlags = nFlag; //设置标记
 ymrParam.Brightness = fValue; //设置亮度值
 ymrParam.Contrast = fValue; //设置对比度值
 ymrParam.Hue = fValue; //设置色调值
 ymrParam.Saturation = fValue; //设置饱和度值
 pControl->SetProcAmpControl(0,&ymrParam); //设置视频图像颜色信息
 }
 }
}

```

(17) 处理“快进”按钮的单击事件，利用 IMediaPosition 接口的 put\_CurrentPosition 方法来设置当前播放的位置，以实现快进功能。其实现代码如下：

```
void CDirectShowEventDlg::OnAddSpeed()
{
 IMediaPosition* pPosition = NULL; //定义IMediaPosition接口指针
 if (pGraph != NULL)
 {
 pGraph->QueryInterface(IID_IMediaPosition, (void**)&pPosition); //获取IMediaPosition接口对象
 if (pPosition != NULL)
 {
 REFTIME curTime, endTime;
 pPosition->get_StopTime(&endTime); //获取播放的停止时间
 pPosition->get_CurrentPosition(&curTime); //获取当前的播放时间
 curTime += 5; //设置快进
 if (curTime <= endTime)
 {
 pPosition->put_CurrentPosition(curTime); //设置当前播放的时间
 }
 else
 {
 pPosition->put_CurrentPosition(endTime); //设置播放时间为停止时间
 }
 }
 }
}
```

(18) 处理“增大音量”按钮的单击事件，利用 IBasicAudio 接口的 put\_Volume 方法来调节音量。其实现代码如下：

```
void CDirectShowEventDlg::OnVolumnmax()
{
 IBasicAudio* pAudio = NULL; //定义IBasicAudio接口指针
 if (pGraph != NULL)
 {
 pGraph->QueryInterface(IID_IBasicAudio, (void**)&pAudio); //获取IBasicAudio接口对象
 if (pAudio != NULL) //如果有音频数据
 {
 pAudio->get_Volume(&m_IVolumn); //获取当前的音量
 if (m_IVolumn < 0) //当前不是最大音量
 {
 m_IVolumn += 200; //增加音量
 pAudio->put_Volume(m_IVolumn); //设置音量
 }
 else
 {
 m_IVolumn = 0; //当前音量为最大音量
 }
 }
 }
}
```

(19) 处理“黑白图像”按钮的单击事件，调节视频图像的饱和度为最小值，这样就实现了黑白图像的效果。其实现代码如下：

```
void CDirectShowEventDlg::OnGray()
{
 IVMRMixerControl9* pControl = NULL; //定义IVMRMixerControl9接口指针
 if (pRender != NULL)
 {
 //获取IVMRMixerControl9接口对象
 pRender->QueryInterface(IID_IVMRMixerControl9, (void**)&pControl);
 if (pControl != NULL)
 {
 VMR9ProcAmpControl vmrParam; //定义参数
 memset(&vmrParam, 0, sizeof(VMR9ProcAmpControl)); //初始化参数
 vmrParam.dwSize = sizeof(VMR9ProcAmpControl); //设置参数大小
 vmrParam.dwFlags = ProcAmpControl9_Saturation; //设置参数标记
 //获取饱和度的最小值
 VMR9ProcAmpControlRange range;
 range.dwSize = sizeof(VMR9ProcAmpControlRange);
 range.dwProperty = ProcAmpControl9_Saturation;
 pControl->GetProcAmpControlRange(0, &range);
 //设置饱和度和为最小值
 if (m_bGrayImage == FALSE) //设置黑白图像
 {
 VMR9ProcAmpControl getParam; //定义参数
 memset(&getParam, 0, sizeof(VMR9ProcAmpControl)); //初始化参数
 getParam.dwSize = sizeof(VMR9ProcAmpControl); //设置参数大小
 }
 }
 }
}
```

```

getParam.dwFlags = ProcAmpControl9_Saturation; //设置参数标记
pControl->GetProcAmpControl(0,&getParam); //获取饱和度
m_fSaturation = range.MinValue;
vmrParam.Saturation = m_fSaturation; //设置饱和度和度
m_bGrayImage = TRUE;
m_GrayBtn.SetWindowText("彩色图像");

}
else //设置彩色图像
{
 m_fSaturation = 1;
 m_GrayBtn.SetWindowText("黑白图像");
 vmrParam.Saturation = m_fSaturation;
 m_bGrayImage = FALSE;
}
//设置黑白图像或彩色图像
HRESULT hRet = pControl->SetProcAmpControl(0,&vmrParam);
}
}

```

(20) 创建一个对话框类，类名为“CDisplayWnd”，该类将作为视频显示窗口，显示影片图像。

(21) 向对话框中添加一个图片等控件，设置图片控件的类型为-Bitmap，设置 Image 属性为程序中到导入的位图资源 ID。

(22) 处理 CDisplayWnd 类的 WM\_SIZE 消息，在对话框大小改变时调整视频图像的大小，使其适应窗口的大小。其实现代码如下：

```

void CDisplayWnd::OnSize(UINT nType, int cx, int cy)
{
 CDialog::OnSize(nType, cx, cy);
 CDirectShowEventDlg *pDlg = (CDirectShowEventDlg *)AfxGetMainWnd();
 if (pDlg->m_bViewPlay)
 {
 CRect ClientRC,rc;
 GetClientRect(ClientRC);
 m_Panel.MoveWindow(ClientRC);
 m_Panel.ModifyStyle(SS_BITMAP,SS_BLACKRECT);
 m_Panel.GetClientRect(rc);
 pDlg->pViewWnd->put_Height(rc.Height());
 pDlg->pViewWnd->put_Width(rc.Width());
 }
}

```

(23) 新建一个对话框类，类名为“CVideoSet”，用于设置视频图像的色彩信息，包括视频图像的亮度、饱和度和对比度等。

(24) 处理对话框的 WM\_HSCROLL 消息，在用户拖动滑块时，将执行该消息处理函数，设置视频图像相应的信息。其实现代码如下：

```

void CVideoSet::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
 if (pScrollBar != NULL)
 {
 CDirectShowEventDlg *pMainDlg = NULL;
 //获取对话框主窗口指针
 pMainDlg = (CDirectShowEventDlg *)AfxGetMainWnd();
 pMainDlg->m_bGrayImage = FALSE;
 pMainDlg->m_GrayBtn.SetWindowText("黑白图像"); //设置按钮文本
 pMainDlg->m_GrayBtn.Invalidate(); //更新按钮
 if (pScrollBar->m_hWnd==m_Hue.m_hWnd) //色调滑块的滚动消息
 {
 if (nSBCode==SB_THUMBPOSITION) //拖动滚动块
 {
 m_Hue.SetPos(nPos); //设置滑块位置
 }
 int nCurPos = m_Hue.GetPos(); //获取滑块位置
 CString csPos;
 csPos.Format("%i",nCurPos);
 m_HueNum.SetWindowText(csPos);
 //调用主对话框的SetViewInfo方法设置视频图像的色调
 pMainDlg->SetViewInfo(ProcAmpControl9_Hue,nCurPos);
 pMainDlg->m_fHue = nCurPos;
 }
 else if (pScrollBar->m_hWnd==m_Saturation.m_hWnd) //饱和度滑块的滚动消息
 {

```



```

 if (nSBCode==SB_THUMBPOSITION) //拖动滚动块
 {
 m_Saturation.SetPos(nPos); //设置滑块位置
 }
 int nCurPos = m_Saturation.GetPos(); //获取滑块位置
 CString csPos;
 csPos.Format("%i",nCurPos);
 m_SatNum.SetWindowText(csPos);
 //调用主对话框的SetViewInfo方法设置视频图像的饱和度
 pMainDlg->SetViewInfo(ProcAmpControl9_Saturation,nCurPos/100.0);
 pMainDlg->m_fSaturation = nCurPos/100.0;
 }
 else if (pScrollBar->m_hWnd==m_Brightness.m_hWnd) //亮度滑块的滚动消息
 {
 if (nSBCode==SB_THUMBPOSITION) //拖动滚动块
 {
 m_Brightness.SetPos(nPos); //设置滑块位置
 }
 int nCurPos = m_Brightness.GetPos(); //获取滑块位置
 CString csPos;
 csPos.Format("%i",nCurPos);
 m_BrightNum.SetWindowText(csPos);
 //调用主对话框的SetViewInfo方法设置视频图像的亮度
 pMainDlg->SetViewInfo(ProcAmpControl9_Brightness,nCurPos);
 pMainDlg->m_fBright = nCurPos;
 }
 else if (pScrollBar->m_hWnd==m_Contrast.m_hWnd) //对比度滑块的滚动消息
 {
 if (nSBCode==SB_THUMBPOSITION) //拖动滚动块
 {
 m_Contrast.SetPos(nPos); //设置滑块位置
 }
 int nCurPos = m_Contrast.GetPos(); //获取滑块位置
 CString csPos;
 csPos.Format("%i",nCurPos);
 m_ConNum.SetWindowText(csPos);
 //调用主对话框的SetViewInfo方法设置视频图像的对比度
 pMainDlg->SetViewInfo(ProcAmpControl9_Contrast,nCurPos/100.0);
 pMainDlg->m_fContrast = nCurPos/100.0;
 }
}
CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}

```

(25)在对话框初始化时获取当前视频图像各项信息的默认值,作为视频设置窗口的初始值。

其实现代码如下:

```

BOOL CVideoSet::OnInitDialog()
{
 CDialog::OnInitDialog();
 CDirectShowEventDlg *pMainDlg = NULL;
 //获取对话框主窗口指针
 pMainDlg = (CDirectShowEventDlg *)AfxGetMainWnd();
 if (pMainDlg)
 {
 //设置色调滑块的滚动范围
 m_Hue.SetRange(pMainDlg->m_HueRange.MinValue,pMainDlg->m_HueRange.MaxValue);
 WPARAM wParam;
 MAKEWPARAM(SB_THUMBPOSITION,pMainDlg->m_fHue);
 m_Hue.SetPos(100);
 m_Hue.SetPos(pMainDlg->m_fHue);
 //执行对话框的WM_HSCROLL消息处理函数
 SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Hue.m_hWnd);
 //设置饱和度滑块的滚动范围
 m_Saturation.SetRange(pMainDlg->m_SatRange.MinValue*100,
 pMainDlg->m_SatRange.MaxValue*100);
 MAKEWPARAM(SB_THUMBPOSITION,pMainDlg->m_fSaturation*100);
 m_Saturation.SetPos(100);
 m_Saturation.SetPos(pMainDlg->m_fSaturation*100);
 //执行对话框的WM_HSCROLL消息处理函数
 SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Saturation.m_hWnd);
 //设置亮度滑块的滚动范围
 m_Brightness.SetRange(pMainDlg->m_BrightRange.MinValue,
 pMainDlg->m_BrightRange.MaxValue);
 MAKEWPARAM(SB_THUMBPOSITION,pMainDlg->m_fBright);
 m_Brightness.SetPos(100);
 m_Brightness.SetPos(pMainDlg->m_fBright);
 //执行对话框的WM_HSCROLL消息处理函数
 }
}

```



```

SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Brightness.m_hWnd);
//设置对比度滑块的滚动范围
m_Contrast.SetRange(pMainDlg->m_ConRange.MinValue*100,
 pMainDlg->m_ConRange.MaxValue*100);
MAKEWPARAM(SB_THUMBPOSITION,1*100);
m_Contrast.SetPos(100);
m_Contrast.SetPos(1*100);
//执行对话框的WM_HSCROLL消息处理函数
SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Contrast.m_hWnd);
}
return TRUE;
}

```

(26) 处理“默认”按钮的单击事件，恢复视频图像默认效果。其实现代码如下：

```

void CVideoSet::OnDefault()
{
 CDirectShowEventDlg *pMainDlg = NULL;
 pMainDlg = (CDirectShowEventDlg *)AfxGetMainWnd(); //获取主对话框指针
 if (pMainDlg)
 {
 //设置色调滑块的滚动范围
 m_Hue.SetRange(pMainDlg->m_HueRange.MinValue,pMainDlg->m_HueRange.MaxValue); WPARAM wParam;
 MAKEWPARAM(SB_THUMBPOSITION,1);
 m_Hue.SetPos(100);
 m_Hue.SetPos(1);
 //执行对话框的 WM_HSCROLL消息处理函数
 SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Hue.m_hWnd);
 pMainDlg->m_fHue = 1;
 //设置饱和度滑块的滚动范围
 m_Saturation.SetRange(pMainDlg->m_SatRange.MinValue*100,
 pMainDlg->m_SatRange.MaxValue*100);
 MAKEWPARAM(SB_THUMBPOSITION,1*100);
 m_Saturation.SetPos(100);
 m_Saturation.SetPos(1*100);
 //执行对话框的WM_HSCROLL消息处理函数
 SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Saturation.m_hWnd);
 pMainDlg->m_fSaturation = 1;
 //设置亮度滑块的滚动范围
 m_Brightness.SetRange(pMainDlg->m_BrightRange.MinValue,
 pMainDlg->m_BrightRange.MaxValue);
 MAKEWPARAM(SB_THUMBPOSITION,1);
 m_Brightness.SetPos(100);
 m_Brightness.SetPos(1);
 //执行对话框的WM_HSCROLL消息处理函数
 SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Brightness.m_hWnd);
 pMainDlg->m_fBright = 1;
 //设置对比度滑块的滚动范围
 m_Contrast.SetRange(pMainDlg->m_ConRange.MinValue*100,
 pMainDlg->m_ConRange.MaxValue*100);
 MAKEWPARAM(SB_THUMBPOSITION,1*100);
 m_Contrast.SetPos(100);
 m_Contrast.SetPos(1*100);
 //执行对话框的WM_HSCROLL消息处理函数
 SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Contrast.m_hWnd);
 pMainDlg->m_fContrast = 1;
 }
}

```

(27) 处理“色调”编辑框的文本改变时的事件，当用户在编辑框中输入色调值时将设置滑块显示的位置，这样会触发对话框的 WM\_HSCROLL 消息，最终在 WM\_HSCROLL 消息处理函数中设置色调信息。其实现代码如下：

```

void CVideoSet::OnChangeHueNum()
{
 CString csText;
 m_HueNum.GetWindowText(csText); //获取色调值
 if (!csText.IsEmpty())
 {
 m_Hue.SetPos(atoi(csText)); //设置色调滑块位置
 }
}

```

## 举一反三

根据本实例，读者可以：

- 设计具有记忆功能的 MP3 播放器。

# 第 5 章

## 文件系统

- 文件的基本操作
- 查找文件
- 与文件目录相关的命令操作
- 文件、文件夹的复制和移动
- 文件修改
- 文件的读取与保存
- 文件管理
- 加密与解密
- INI 文件
- 其他

Visual C++

资源分享

PDG

## 5.1 文件的基本操作

创建和删除文件是 Windows 最基本的操作，本节通过几个实例介绍如何利用程序创建和删除文件。

### 实例 173 创建和删除文件夹

本实例可以提高工作效率

实例位置：光盘\mingrisoft\05\173

#### 实例说明

在开发应用程序时，可以通过程序来创建和删除文件夹。本实例主要实现创建和删除文件夹的功能，运行程序，在编辑框中输入文件夹名 ppp，如图 5.1 所示，单击“创建”按钮，就会在程序根目录下创建 ppp 文件夹，单击“删除”按钮，会把程序根目录下的 ppp 文件夹删除。

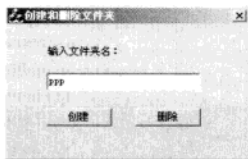


图 5.1 创建和删除文件夹

#### 技术要点

使用 API 函数 CreateDirectory 可以创建文件夹，使用 Get CurrentDirectory 函数可以在一个缓冲区中装载当前目录。

CreateDirectory 函数原型如下：

```
BOOL CreateDirectory(LPCTSTR lpPathName, LPSECURITY_ATTRIBUTES lpSecurityAttributes);
```

参数说明：

- lpPathName：目录的名字。
- lpSecurityAttributes：这个结构定义了目录的安全特性。

GetCurrentDirectory 函数原型如下：

```
DWORD GetCurrentDirectory(DWORD nBufferLength, LPTSTR lpBuffer);
```

参数说明：

- nBufferLength：缓冲区的长度。
- lpBuffer：指定一个预定义字符串，用于装载当前目录。

#### 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“创建和删除文件夹”。
- (2) 向窗体中添加一个文本编辑框控件和两个按钮控件。
- (3) 主要程序代码。

```
//创建文件夹
void CCreateFolderDlg::OnButcreate()
{
 // TODO: Add your control notification handler code here
 char buf[256];
 ::GetCurrentDirectory(256,buf); //获取程序根目录路径
 m_name.GetWindowText(name);
 strcat(buf,"\\");
 strcat(buf,name);
 if(CreateDirectory(buf,NULL))//创建目录
 {
 MessageBox("文件夹创建成功!");
 return;
 }
}

//删除文件夹
void CCreateFolderDlg::OnButdel()
{
 // TODO: Add your control notification handler code here
 char buf[256];
 ::GetCurrentDirectory(256,buf); //获取当前目录
```

```

m_name.GetWindowText(name);
strcat(buf, "\\");
strcat(buf, name);
if(RemoveDirectory(buf))//删除目录
{
 MessageBox("文件夹删除成功!");
 return;
}
}

```

### 举一反三

根据本实例，读者可以：

- 创建和删除指定目录下文件夹。

## 实例 174 把文件删除到回收站中

本实例可以提高工作效率

实例位置：光盘\mingrisoft\05\174

### 实例说明

在应用程序中有时需要删除一些外部文件，如程序运行时建立的资料文件等。删除文件的方法也多种多样，不过大部分都是将文件直接删除，这种方法比较简单。不过当出现误操作时删除的文件便很难恢复了。本例提供的方法是将文件删除到回收站中，如果又不想删除该文件，则可以到回收站中将其复原。运行程序，结果如图 5.2 所示。单击“浏览”按钮选择要删除的文件，单击“删除”按钮进行删除。

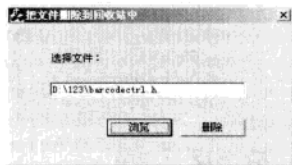


图 5.2 把文件删除到回收站中

### 技术要点

SHFileOperation 是 Windows 提供的对文件系统对象进行删除、移动和复制等操作的 API 函数，该函数原型如下：

```
WINHELLAPI int WINAPI SHFileOperation(LPSHFILEOPSTRUCT lpFileOp);
```

参数说明：

- lpFileOp：是一个指向 SHFILEOPSTRUCT 结构的指针，可以通过设置该结构中的变量来控制对文件的操作。该结构定义如下：

```

typedef struct _SHFILEOPSTRUCT{
 HWND hwnd;
 UINT wFunc;
 LPCSTR pFrom;
 LPCSTR pTo;
 FILEOP_FLAGS fFlags;
 BOOL fAnyOperationsAborted;
 LPVOID hNameMappings;
 LPCSTR lpProgressTitle;
} SHFILEOPSTRUCT, FAR *LPSHFILEOPSTRUCT;

```

- hwnd：拥有者窗口句柄。
- wFunc：文件操作功能，可选值如表 5.1 所示。

表 5.1 wFunc 参数可选值

| 参 数       | 描 述                            |
|-----------|--------------------------------|
| FO_COPY   | 复制 pFrom 指定的文件到 pTo 指定的位置      |
| FO_DELETE | 删除 pFrom 指定的文件                 |
| FO_MOVE   | 移动 pFrom 指定的文件到 pTo 指定的位置      |
| FO_RENAME | 用 pTo 指定的新文件名替换 pFrom 指定文件的文件名 |

- pFrom：源文件。



- pTo: 目标文件。
- fFlags: 文件控制标志。
- fAnyOperationsAborted: 用户是否中断操作。
- hNameMappings: 指向一个 SHNAMEMAPPING 结构的指针。
- lpszProgressTitle: 进程标题。

## 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“把文件删除到回收站中”。
- (2) 在窗体上添加一个文本编辑框控件和两个按钮控件。
- (3) 主要程序代码。

```
void CFileInCallbackDlg::OnButdelete()
{
 // TODO: Add your control notification handler code here
 char fileName[100]="0";
 strcpy(fileName,strText);
 strcat(fileName,"0");
 SHFILEOPSTRUCT shfile;
 shfile.hwnd = 0;
 shfile.wFunc = FO_DELETE;
 shfile.pFrom = fileName;
 shfile.pTo = NULL;
 shfile.fFlags = FOF_ALLOWUNDO;
 shfile.hNameMappings = NULL;
 shfile.lpszProgressTitle = NULL;
 SHFileOperation(&shfile);
}
```

## 举一反三

根据本实例，读者可以：

- 定时删除文件夹；
- 复制文件。

## 实例 175 清空回收站

本实例可以提高工作效率

实例位置：光盘\mingrisoft\05\175

## 实例说明

“回收站”顾名思义是用来存放垃圾的。在 Windows 系统中，回收站是一个存放已删除文件的地方。其实回收站是一个系统文件夹，在 DOS 模式下进入磁盘根目录，键入“dir/a”则会看到一个名为 RECYCLED 的文件夹，这个就是回收站。为了防止误删除操作，Windows 将用户删除的文件先暂存到回收站中，使删除的文件可以恢复，待确认删除后再将回收站清空。本例将编程完成这个清空回收站的工作，为用户清理出一些磁盘空间。运行程序，如图 5.3 所示，单击“清空回收站”按钮，就可以清空回收站了。

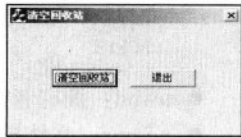


图 5.3 清空回收站

## 技术要点

本例使用 SHEmptyRecycleBin 函数来清空回收站。SHEmptyRecycleBin 函数原型如下：

SHSTDAPI SHEmptyRecycleBin( HWND hwnd, LPCTSTR pszRootPath, DWORD dwFlags );

参数说明：

- hwnd: 父窗口句柄。
- pszRootPath: 设置删除哪个磁盘的回收站，如果设置字符串为空则清空所有的回收站。

- dwFlags: 用于清空回收站的功能参数。

## 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“清空回收站”。
- (2) 向窗体中添加二个按钮控件。
- (3) 主要程序代码。

```
void CClearhszDlg::OnButclear()
{
 // TODO: Add your control notification handler code here
 GetWindowLong(m_hWnd, 0);
 SHEmptyRecycleBin(m_hWnd, NULL, SHERB_NOCONFIRMATION || SHERB_NOPROGRESSUI || SHERB_NOSOUND);
 MessageBox("回收站已清空!");
}
```

## 举一反三

根据本实例，读者可以：

- 定时清空回收站。

## 实例 176 强制删除文件

本实例可以提高工作效率

实例位置：光盘\mingrisoft\05\176

## 实例说明

强制删除文件是指当用户指定了要删除的文件路径时，首先查看系统中所打开的进程中是否有当前所要删除的文件。如果文件被打开，将关闭打开进程后再将文件删除。如果不关闭进程直接删除文件，文件删除操作将失败。如图 5.4 所示。

## 技术要点

在该实例中使用 NtQuerySystemInformation 函数来获取当前系统中所运行的所有进程。函数原型如下：

```
typedef DWORD (WINAPI *PNTQuerySystemInformation)(DWORD, VOID*, DWORD, ULONG*);
```

该函数接收 4 个参数，第 1 个参数是该函数所使用的类型信息；第 2 个参数是一个结构指针，该结构存储了通过该函数所获取的进程信息；第 3 个参数是结构的大小；第 4 个参数是返回结构信息的大小，可设为 NULL。

获取进程信息所需要的结构定义如下：

```
typedef struct _SYSTEM_PROCESS_INFORMATION
{
 DWORD dNext; //构成结构序列的偏移量
 DWORD dThreadCount; //线程数目
 DWORD dReserved01;
 DWORD dReserved02;
 DWORD dReserved03;
 DWORD dReserved04;
 DWORD dReserved05;
 DWORD dReserved06;
 QWORD qCreateTime; //创建时间
 QWORD qUserTime; //用户模式(Ring 3)的CPU时间
 QWORD qKernelTime; //内核模式(Ring 0)的CPU时间
 UNICODE_STRING usName; //进程名称
 DWORD BasePriority; //进程优先级
 DWORD dUniqueProcessId; //进程标识符
 DWORD dInheritedFromUniqueProcessId; //父进程的标识符
 DWORD dHandleCount; //句柄数目
 DWORD dReserved07;
}
```

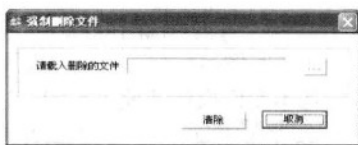


图 5.4 强制删除文件

```
DWORD dReserved08;
VM_COUNTERS VmCounters; //虚拟存储器的结构
DWORD dCommitCharge;
SYSTEM_THREAD Threads[1]; //进程相关线程的结构数组
} SYSTEM_PROCESS_INFORMATION;
```

```
typedef struct _VM_COUNTERS
{
 DWORD PeakVirtualSize; //虚拟存储峰值大小
 DWORD VirtualSize; //虚拟存储大小
 DWORD PageFaultCount; //页故障数目
 DWORD PeakWorkingSetSize; //工作集峰值大小
 DWORD WorkingSetSize; //工作集大小
 DWORD QuotaPeakPagedPoolUsage; //分页池使用配额峰值
 DWORD QuotaPagedPoolUsage; //分页池使用配额
 DWORD QuotaPeakNonPagedPoolUsage; //非分页池使用配额峰值
 DWORD QuotaNonPagedPoolUsage; //非分页池使用配额
 DWORD PagefileUsage; //页文件使用情况
 DWORD PeakPagefileUsage; //页文件使用峰值
} VM_COUNTERS;
```

```
typedef struct _SYSTEM_THREAD
{
 DWORD u1;
 DWORD u2;
 DWORD u3;
 DWORD u4;
 DWORD ProcessId;
 DWORD ThreadId;
 DWORD dPriority;
 DWORD dBasePriority;
 DWORD dContextSwitches;
 DWORD dThreadState; // 2=running, 5=waiting
 DWORD WaitReason;
 DWORD u5;
 DWORD u6;
 DWORD u7;
 DWORD u8;
 DWORD u9;
} SYSTEM_THREAD;
```

在本实例运行前，首先应设置当前进程应具有管理权限，并将进程的权限提升到支持调试，否则是不允许操作其他进程的。这一功能由方法 AdjustPrivilege 来实现。代码如下：

```
//调整进程特权，使其具有DEBUG特权
BOOL CForceDeleteDlg::AdjustPrivilege()
{
 HANDLE hToken = NULL;
 LUID ulID;
 TOKEN_PRIVILEGES privValue;
 //打开进程访问标识
 BOOL bPenod = OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken);
 if (!bPenod)
 {
 return FALSE;
 }
 //查找特权值
 BOOL bLooked = LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &ulID);
 if (!bLooked)
 {
 CloseHandle(hToken);
 return FALSE;
 }

 privValue.PrivilegeCount = 1;
 privValue.Privileges[0].Luid = ulID;
 privValue.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
 BOOL bAdjusted = AdjustTokenPrivileges(hToken, FALSE, &privValue, sizeof(privValue), NULL, NULL);
 if (!bAdjusted)
 {
 CloseHandle(hToken); //关闭句柄
 return FALSE;
 }

 CloseHandle(hToken);
 return TRUE;
}
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个文本编辑框控件，用来显示删除的文件路径。添加两个按钮控件，分别设置其 Caption 属性为“清除”和“取消”。
- (3) 在窗体类中实现 EnumRunningProc 方法，该方法将获取当前系统中运行的所有进程的信息，并将这些信息按进程 ID 和进程结构成对的添加到 pSysProcess 列表中。实现代码如下：

```
//列举系统运行的进程信息
void CForceDeleteDlg::EnumRunningProc()
{
 //列举打开的对象句柄
 int nCount = 0;
 SYSTEM_PROCESS_INFORMATION* pSysProcess = NULL;
 //获取系统中进程信息第一个结构
 DWORD dwRet = NtQuerySystemInformation(5, m_pProcInfo, m_dwProcSize, NULL);
 if (dwRet != 0)
 {
 return;
 }
 m_SysProcesses.RemoveAll();
 pSysProcess = (SYSTEM_PROCESS_INFORMATION*)m_pProcInfo;
 do
 {
 //添加到进程信息列表
 m_SysProcesses.SetAt(pSysProcess->UniqueProcessId, pSysProcess);
 if (pSysProcess->Next != 0)
 {
 //获取下一个进程信息
 pSysProcess = (SYSTEM_PROCESS_INFORMATION*)((UCHAR*)pSysProcess + pSysProcess->Next);
 }
 else
 {
 pSysProcess = NULL;
 }
 }
 while (pSysProcess != NULL);
}
```

- (4) 在窗体类中实现 EnumProcessOpenedFile 方法，该方法用来获取系统中所有线程句柄，并判断是否为真正的文件句柄，如果是文件句柄将其添加到 m\_FileHandles 文件句柄列表中。实现代码如下：

```
//列举进程打开的文件
void CForceDeleteDlg::EnumProcessOpenedFile()
{
 //列举打开的对象句柄
 int nCount = 0;
 DWORD dwBufferSize = 0x200;
 DWORD dwFactSize = 0;
 //在当前进程中为SYSTEM_HANDLE_INFORMATION结构创建存储空间
 SYSTEM_HANDLE_INFORMATION* pSysHandleInformation = (SYSTEM_HANDLE_INFORMATION*)
 VirtualAlloc(NULL, dwBufferSize, MEM_COMMIT, PAGE_READWRITE);
 //获取SYSTEM_HANDLE_INFORMATION结构信息的大小
 NtQuerySystemInformation(16, pSysHandleInformation, dwBufferSize, &dwFactSize);
 VirtualFree(pSysHandleInformation, 0, MEM_RELEASE);
 //分配实际空间大小
 pSysHandleInformation = (SYSTEM_HANDLE_INFORMATION*)
 VirtualAlloc(NULL, dwBufferSize = dwFactSize + 256, MEM_COMMIT, PAGE_READWRITE);
 //获取系统中所有线程句柄信息
 NtQuerySystemInformation(16, pSysHandleInformation, dwBufferSize, NULL);
 //遍历句柄对象
 int nCurrentID = GetCurrentProcessId();
 //清除列表中的文件句柄
 m_FileHandles.RemoveAll();
 for(int i=0; i<pSysHandleInformation->Count; i++)
 {
 int nProcessID = pSysHandleInformation->Handles[i].ProcessID; //进程ID
 HANDLE hDup = (HANDLE)pSysHandleInformation->Handles[i].HandleNumber;
 if (nProcessID != nCurrentID) //远程进程
 }
```



```

 {
 HANDLE hProcess = NULL;
 hProcess = ::OpenProcess(PROCESS_DUP_HANDLE, TRUE, nProcessID); //打开进程
 HANDLE hFile = (HANDLE)pSysHandleInformation->Handles[i].HandleNumber;
 //复制远程进程句柄到当前进程中
 DuplicateHandle(hProcess, hFile, GetCurrentProcess(), &hDup, 0, FALSE, DUPLICATE_SAME_ACCESS);
 CloseHandle(hProcess);

 //判断是否为文件句柄
 int nSize = 0;
 NtQueryObject(hDup, 2, NULL, 0, &nSize);
 UCHAR *lpBuffer = new UCHAR[nSize];
 NtQueryObject(hDup, 2, lpBuffer, nSize, NULL);
 UCHAR* pTmp = lpBuffer;
 pTmp += 0x60;
 if (wcsncmp((unsigned short*)pTmp, L"File") == 0)
 {
 //进一步判断当前句柄是否为真正的文件句柄
 HANDLE hTmp = CreateFileMapping(hDup, NULL, PAGE_READONLY, 0, 20, NULL);
 DWORD dwError = GetLastError();
 //dwError == 8, 错误信息是存储空间不足, 无法处理此命令, 依据此信息, 我们认为文件已存在
 if (hTmp != NULL || dwError == 8) //判断是否为文件句柄
 {
 m_FileHandles.AddTail(pSysHandleInformation->Handles[i]);
 CloseHandle(hTmp);
 }
 }
 CloseHandle(hDup);
 delete [] lpBuffer;
 }
}
VirtualFree(pSysHandleInformation, 0, MEM_RELEASE);
}

```

(5) 在窗体类中实现 CloseFileHandle 方法, 该方法利用远程线程注入的方法在文件所在进程将文件句柄关闭。实现代码如下:

```

//在远程线程中关闭文件句柄
BOOL CForceDeleteDlg::CloseFileHandle(DWORD dwProcessID, HANDLE hFile)
{
 //根据进程ID打开进程, 获取进程句柄
 HANDLE hProcess =
 OpenProcess(PROCESS_CREATE_THREAD|PROCESS_VM_OPERATION|PROCESS_VM_WRITE|PROCESS_VM_READ,
 FALSE, dwProcessID);

 if (hProcess == NULL)
 {
 return FALSE;
 }

 HMODULE hMod = LoadLibrary("kernel32.dll");

 for (POSITION pos = m_FileHandles.GetHeadPosition(); pos != NULL;)
 {
 SYSTEM_HANDLE& fileHandle = m_FileHandles.GetNext(pos);

 //在进程中查找句柄
 SYSTEM_PROCESS_INFORMATION* pProcInfo = NULL;
 BOOL bFindRet = m_SysProcesses.Lookup(dwProcessID, pProcInfo);
 if (!bFindRet)
 {
 continue;
 }

 HANDLE hFile = (HANDLE)fileHandle.HandleNumber;

 DWORD dwThreadId = 0;
 HANDLE hThread;
 //创建远程线程
 hThread = CreateRemoteThread(hProcess, NULL, 0,
 (DWORD(_stdcall*)(void*))GetProcAddress(hMod, "CloseHandle"),
 hFile, 0, &dwThreadId);

 if (hThread == NULL)
 {
 CloseHandle(hProcess);
 FreeLibrary(hMod); //释放动态库
 return FALSE;
 }
 }
}

```

```

 }

 CloseHandle(hProcess);
 FreeLibrary(hMod);
 return TRUE;
}

```

(6) 在窗体中设置好需要删除的文件后,单击“清除”按钮实现对文件的删除操作。在删除时会判断该文件是可执行文件还是由其他进程运行的文件,并进行处理。实现代码如下:

```

void CForceDeleteDlg::OnClear()
{
 //列举运行中的进程
 EnumRunningProc();
 //列举进程打开的文件句柄
 EnumProcessOpenedFile();
 //遍历文件句柄

 m_bFindFile = FALSE;
 DWORD dwCurProcID = GetCurrentProcessId();
 int nCount = 0;
 CString szFileName, szDeviceName;
 m_FileName.GetWindowText(szFileName);
 GetDeviceName(szFileName, szDeviceName); //获取驱动器名称

 for (POSITION pos = m_FileHandles.GetHeadPosition(); pos != NULL;)
 {
 SYSTEM_HANDLE& fileHandle = m_FileHandles.GetNext(pos);

 //在进程中查找句柄
 SYSTEM_PROCESS_INFORMATION* pProcInfo = NULL;

 BOOL bFindRet = m_SysProcesses.Lookup(fileHandle.ProcessID, pProcInfo);
 if (!bFindRet)
 {
 continue;
 }
 m_hFile = (HANDLE)fileHandle.HandleNumber;

 if (dwCurProcID != fileHandle.ProcessID)
 {
 HANDLE hProcess = OpenProcess(PROCESS_DUP_HANDLE, TRUE, fileHandle.ProcessID);
 HANDLE hSysFile = (HANDLE)fileHandle.HandleNumber;
 //复制远程进程句柄到当前进程中
 DuplicateHandle(hProcess, hSysFile, GetCurrentProcess(), &m_hFile, 0, FALSE,
 DUPLICATE_SAME_ACCESS);
 CloseHandle(hProcess);
 }

 memset(m_FileNames, 0, MAX_FILENAME);
 NtQueryObject(m_hFile, 1, m_FileNames, MAX_FILENAME, NULL);
 CloseHandle(m_hFile);
 //判断文件是否是欲删除的文件
 //首先将DOS文件名转换为设备文件名
 UCHAR* pchData = m_FileNames;
 pchData += 8;
 if (wcsncmp((unsigned short*)pchData, szDeviceName.AllocSysString()) == 0)
 {
 //关闭文件句柄
 CloseFileHandle(fileHandle.ProcessID, (HANDLE)fileHandle.HandleNumber);
 m_bFindFile = TRUE;
 // CloseHandle(m_hFile);
 }
 szDeviceName.ReleaseBuffer();
 nCount++;
 }
 if (m_bFindFile == FALSE)
 {
 //判断文件是否是一个直接运行的文件
 for (POSITION ps = m_SysProcesses.GetStartPosition(); ps != NULL;)
 {
 DWORD dwProcID;
 SYSTEM_PROCESS_INFORMATION* pProcInfo = NULL;
 m_SysProcesses.GetNextAssoc(ps, dwProcID, pProcInfo);

```

```

if (pProcInfo != NULL)
{
 HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, TRUE, dwProcID);
 if (hProcess)
 {
 char chFileName[MAX_PATH] = {0};
 GetModuleFileNameEx(hProcess, NULL, chFileName, MAX_PATH);
 //发现系统中运行的进程
 if (strcmp(szFileName, chFileName) == 0)
 {
 TerminateProcess(hProcess, 0);
 m_bFindFile = true;
 }
 CloseHandle(hProcess);
 }
}
}
}
if (m_bFindFile == TRUE && MessageBox("确实要删除文件吗?", "提示", MB_YESNO) == IDYES)
{
 //删除文件
 BOOL bDeleted = DeleteFile(szFileName);
 if (bDeleted)
 {
 m_FileName.SetWindowText("");
 MessageBox("删除成功!");
 }
}
}
}

```

### 举一反三

- 根据本实例，读者可以：
- 定时清空回收站。

## 5.2 查找文件

本节通过查找文件，检查文件是否存在，提取指定文件夹目录到 INI 文件等实例介绍文件的查找的思路及相关算法。

### 实例 177 搜索文件

本实例可以方便操作、提高效率

实例位置：光盘\mingrisoft\05\177

#### 实例说明

本实例实现了在选定的文件夹下查找指定文件的功能。运行程序，通过“查找”按钮可以将查找指定文件后的结果显示在列表控件中，如图 5-5 所示。

#### 技术要点

用户自定义函数 FindFile 来实现查找功能，该函数是递归函数，通过 CFileFind 对象查找文件。CFileFind 对象的方法有很多，主要有 FindFile、FindNextFile、IsDots、IsDirectory 和 GetFileName。下面介绍这几个方法。

(1) FindFile 方法用来查找指定文件。原型如下：

```
virtual BOOL FindFile(LPCWSTR pstrName = NULL, DWORD dwUnused = 0);
```

参数说明：

- pstrName：指向文件名的字符串指针。

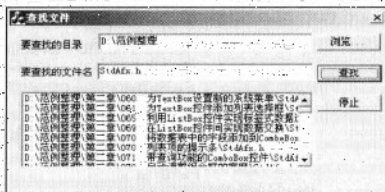


图 5-5 搜索文件

● dwUnused: 固定值, 该值为零。

(2) FindNextFile 方法用于查找下一个文件, 通过返回值可以判断是否是查找的文件。

(3) IsDots 方法用于判断目标文件是否是“.”或“..”。

(4) IsDirectory 方法用于判断目标文件是否是文件夹。

(5) GetFileName 方法用于获得查找到文件的文件名。

## 实现过程

(1) 新建名为 FindFile 的对话框 MFC 工程。

(2) 在对话框上添加 2 个静态文本控件; 添加 2 个文本编辑框控件, ID 属性分别设置为 IDC\_EDADD 和 IDC\_EDFILENAME; 添加 3 个按钮控件, ID 属性分别设置为 IDC\_ADD、IDC\_FIND 和 IDC\_BTSTOP, Caption 属性分别设置为“浏览...”、“查找”和“停止”。

(3) 在头文件 FindFileDlg.h 加入如下变量声明:

```
CString strfilename;
BOOL bstop;
```

(4) 在 OnInitDialog 中设置初始变量, 代码如下:

```
BOOL CFindFileDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 bstop=FALSE;
 return TRUE;
}
```

(5) “浏览...”按钮的实现函数, 该函数用于添加要查找文件所在的文件夹, 代码如下:

```
void CFindFileDlg::OnAdd()
{
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer, MAX_PATH);
 bi.hwndOwner=GetSafeHwnd(); //获取窗口句柄
 bi.pidlRoot=NULL;
 bi.pszDisplayName=buffer;
 bi.lpszTitle="选择一个文件夹";
 bi.ulFlags=BIF_EDITBOX;
 bi.lpfnc=NULL;
 bi.lParam=0;
 bi.iImage=0;
 LPITEMIDLIST pList=NULL;
 if((pList=SHBrowseForFolder(&bi))!=NULL) //显示浏览窗口
 {
 char path[MAX_PATH];
 ZeroMemory(path, MAX_PATH);
 SHGetPathFromIDList(pList, path);
 GetDlgItem(IDC_EDADD)->SetWindowText(path);
 }
}
```

(6) 自定义函数实现查找文件, 代码如下:

```
void CFindFileDlg::FindFile(CString strPath)
{
 CString strtemp;
 if(strPath.Right(1)!="\\")
 strtemp.Format("%s*.*", strPath);
 else
 strtemp.Format("%s*.*", strPath);
 CFileFind findfile;
 BOOL bfind=findfile.FindFile(strtemp); //查找文件
 while(bfind)
 {
 bfind=findfile.FindNextFile(); //查找下一个文件
 if(strfilename==findfile.GetFileName())
 {
 m_filelist.AddString(findfile.GetFilePath());
 }
 if(findfile.IsDirectory()&&!findfile.IsDots())
 {
 FindFile(findfile.GetFilePath());
 }
 }
}
```



```

 }
 if(bstop)return;
 }
}

```

(7) “查找”按钮的实现函数，该函数用于查找文件，代码如下：

```
void CFindFileDialog::OnFind()
{
 CString strpath;
 GetDlgItem(IDC_EDFILENAME)->GetWindowText(strfilename);
 GetDlgItem(IDC_EDADD)->GetWindowText(strpath);
 FindFile(strpath);
 bstop=FALSE;
}
```

(8) “停止”按钮的实现函数，终止对文件的查找，代码如下：

```
void CFindFileDialog::OnBtstop()
{
 bstop=TRUE;
}
```


### 举一反三

根据本实例，读者可以：

- 开发全盘查找程序;
- 查找指定类型的文件。

### 实例 178

## 实例 178 使用多线程实现文件快速搜索

|                                                                                   |                            |
|-----------------------------------------------------------------------------------|----------------------------|
|  | 本实例可以提高工作效率                |
|                                                                                   | 实例位置: 光盘\mingrisoft\05\178 |

实例位置: 光盘\mingrisoft\05\178

## 实例说明

本实例实现了在选中的单个或多个目录下搜索指定的文件的功能。运行程序在窗体左侧的目录列表中选择要搜索的目录，然后在文件名编辑框中输入要搜索的文件名，单击“查找”按钮执行文件搜索操作，并将搜索结果显示在列表控件中，效果如图 5.6 所示。



图 5.6 使用多线程实现文件快速搜索

## 技术要点

在本实例中由于“查找”按钮并不在应用程序的主窗口中,而查找方法定义在程序的主窗口类中,所以需要定义自定义消息实现查找方法的调用。自定义消息如下:

```
#define WMFINDFILE (WM_USER + 10)
```

单击“查找”按钮时向主窗体发送自定义消息，实现代码如下：

```
void CChildFrame::OnFindFile()
{
 m_list.DeleteAllItems();//删除列表中所有项
 GetParent()->GetParent()->
 SendMessage(WM_FINDFILE,(LPARAM)&m_edit,(LPARAM)&m_list);
}
```

在程序的主窗体中需要定义一个消息映射函数用于接收此消息，实现代码如下：

```
ON_MESSAGE(WMFINDFILE, OnFindFile)
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 添加一个新的窗体，修改窗体的 ID 为 IDD\_ChildFrame；添加一个静态文本控件，设

置 Caption 属性为“文件名称”；添加一个文本编辑框控件；添加一个按钮控件，设置 Caption 属性为“查找”；添加一个列表视图控件。

### (3) 主要程序代码。

在主窗体初始化时创建并分割窗体视图，实现代码如下：

```
m_Splitter.CreateStatic(this,1,2);
m_Splitter.CreateView(0,0,RUNTIME_CLASS(CTreeFrame),CSize(150,0),NULL);
m_Splitter.CreateView(0,1,RUNTIME_CLASS(CChildFrame),CSize(0,0),NULL);
CalcSplitterSize();
```

### (4) 实现文件查找消息事件，开启一个线程执行文件查找操作，实现代码如下：

```
LRESULT CFileFindThreadDlg::OnFindFile(WPARAM wParam, LPARAM lParam)
```

```
{
 CTreeCtrl &treeCtrl = ((CTreeFrame *)m_Splitter.GetPane(0,0))>GetTreeCtrl();
 m_strArrPath.RemoveAll();
 GetSearchingPath(treeCtrl.GetRootItem(),&treeCtrl);
```

```
THREADPARAM threadParam;
```

```
CEdit *edit = (CEdit *)(wParam);
edit->GetWindowText(threadParam.FileName,255);
```

```
CListCtrl *listCtrl = (CListCtrl *)lParam;
```

```
threadParam.dlg = this;
threadParam.listCtrl = listCtrl;
//创建并运行线程，并开始查找文件
CWinThread* pThread = AfxBeginThread(
 FindFileThreadFunc,
 & threadParam,
 THREAD_PRIORITY_NORMAL,
 0,
 CREATE_SUSPENDED);
```

```
ASSERT_VALID(pThread);
pThread->ResumeThread();//启动线程
```

```
return (LRESULT)0;
```

### (5) 实现线程调用方法对选中的目录进行搜索并将搜索结果显示在列表控件中，代码如下：

```
UINT FindFileThreadFunc(LPVOID lParam)
```

```
{
 THREADPARAM threadParam = *(THREADPARAM*) lParam;
 int count = threadParam.dlg->m_strArrPath.GetSize();
```

```
for (int i = 0 ; i < count ; i++)
```

```
{
 threadParam.dlg->FindFiles(threadParam.dlg->m_strArrPath.GetAt(i),
 threadParam.FileName,threadParam.listCtrl);//查找文件
}
```

```
AfxMessageBox("查找完成");
return 0;
```

```
void CFileFindThreadDlg::FindFiles(CString strPath, CString strFileName ,CListCtrl *listCtrl)
```

```
{
 if (strPath.Right(1) != _T("\\\"))
 strPath += _T("\\\");
```

```
strPath += _T("*.\");
```

```
CFileFind file;
```

```
BOOL bContinue = file.FindFile(strPath);//查找第一个文件
```

```
while (bContinue)
```

```
{
 bContinue = file.FindNextFile();//查找下一个文件
 if (file.IsDirectory() && ! file.IsDots())//如果是目录，继续查找
 {
 FindFiles(file.GetFilePath(), strFileName ,listCtrl);
 }
 else if (! file.IsDots() && ! file.IsDirectory())//如果是文件
 {
 if (file.GetFileName().Find(strFileName) != -1)
 {

```

```
int index = listCtrl->GetItemCount();
listCtrl->InsertItem(index, file.GetFileName());
listCtrl->SetItemText(index, 1, file.GetFilePath());
```

### 举一反三

根据本实例，读者可以：

- 实现多线程文件清除。

## 实例 179 检查文件是否存在

本实例是一个方便操作、提高效率的程序

实例位置：光盘\mingrisoft\05\179

### 实例说明

在使用文件时不仅要知道文件路径，更重要的是知道这个文件是否存在，如果不存在，程序调用文件时会发生错误。本实例可以检查文件是否存在。运行程序，在编辑框中输入文件所在路径，单击“确定”按钮，弹出判断文件是否存在消息框，效果如图 5.7 所示。

### 技术要点

本例利用 open 函数打开要判断的文件，通过返回的文件句柄判断文件是否存在，open 函数原型如下：

```
int _open(const char *filename, int oflag [, int pmode]);
```

参数说明：

- filename：文件名或文件路径。
- oflag：文件打开模式。

### 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“检查文件是否存在”。
- (2) 在窗体上添加一个文本编辑框控件、一个按钮控件和两个单选按钮控件。
- (3) 主要程序代码。

```
void CFileExistDlg::OnButok()
{
 // TODO: Add your control notification handler code here
 UpdateData(true);
 int i;

 if ((i = open(m_edit, _O_RDONLY)) < 0)
 {
 MessageBox("文件不存在");
 return;
 }
 else
 {
 MessageBox("文件已存在");
 return;
 }
 _lclose(i);
}
```

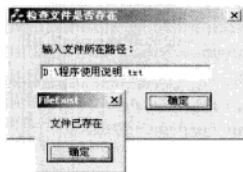


图 5.7 检查文件是否存在

### 举一反三

根据本实例，读者可以：

● 打开并编辑文件。

## 实例 180

提取指定文件夹目录到  
INI 文件

本实例是一个方便操作、提高效率的程序

实例位置：光盘\mingrisoft\05\180

## 实例说明

如果计算机上的文件和文件夹较多，且经常使用，可以将计算机上的文件及文件夹提取出来，以提高效率。手工进行提取这些信息是非常费时费力的，通过本例可以很方便地将文件或文件夹目录信息提取到 INI 文件当中。运行程序，单击“选择文件夹”按钮，选择要提取的文件夹所在的路径，单击“提取目录”按钮将信息提取到 INI 文件当中，然后显示出来，如图 5.8 所示。

## 技术要点

本例通过 CreateFile 函数创建一个 INI 文件，然后利用 WritePrivateProfileString 函数向 INI 文件中写入数据，CreateFile 函数的原型如下：

```
HANDLE CreateFile(
 LPCTSTR lpFileName,
 DWORD dwDesiredAccess,
 DWORD dwShareMode,
 LPSECURITY_ATTRIBUTES lpSecurityAttributes,
 DWORD dwCreationDisposition,
 DWORD dwFlagsAndAttributes,
 HANDLE hTemplateFile
);
```

参数说明：

- lpFileName：文件名。
- dwDesiredAccess：如果为 GENERIC\_READ 表示允许对设备进行读访问；如果为 GENERIC\_WRITE 表示允许对设备进行写访问（可组合使用）；如果为零，表示只允许获取与一个设备有关的信息。
- dwShareMode：零表示不共享；FILE\_SHARE\_READ 和/或 FILE\_SHARE\_WRITE 表示允许对文件进行共享访问。
- lpSecurityAttributes：SECURITY\_ATTRIBUTES 结构的指针，定义了文件的安全特性。
- dwCreationDisposition：可选值如表 5.2 所示。

表 5.2

dwCreationDisposition 参数可选值

| 参 数               | 描 述              |
|-------------------|------------------|
| CREATE_NEW        | 创建文件，如文件存在则会出错   |
| CREATE_ALWAYS     | 创建文件，会改写前一个文件    |
| OPEN_EXISTING     | 文件必须已经存在，由设备提出要求 |
| OPEN_ALWAYS       | 如文件不存在则创建它       |
| TRUNCATE_EXISTING | 将现有文件缩短为零长度      |

- dwFlagsAndAttributes：其值如表 5.3 所示。



图 5.8 提取指定文件夹目录  
到 INI 文件



表 5.3

dwFlagsAndAttributes 参数值

| 参 数                       | 描 述                          |
|---------------------------|------------------------------|
| FILE_ATTRIBUTE_ARCHIVE    | 标记归档属性                       |
| FILE_ATTRIBUTE_COMPRESSED | 将文件标记为已压缩，或者标记为文件在目录中的默认压缩方式 |
| FILE_ATTRIBUTE_NORMAL     | 默认属性                         |
| FILE_ATTRIBUTE_HIDDEN     | 隐藏文件或目录                      |
| FILE_ATTRIBUTE_READONLY   | 文件为只读                        |
| FILE_ATTRIBUTE_SYSTEM     | 文件为系统文件                      |
| FILE_FLAG_WRITE_THROUGH   | 操作系统不得推迟对文件的写操作              |
| FILE_FLAG_OVERLAPPED      | 允许对文件进行重叠操作                  |
| FILE_FLAG_NO_BUFFERING    | 禁止对文件进行缓冲处理。文件只能写入磁盘卷的扇区块    |
| FILE_FLAG_RANDOM_ACCESS   | 针对随机访问对文件缓冲进行优化              |
| FILE_FLAG_SEQUENTIAL_SCAN | 针对连续访问对文件缓冲进行优化              |
| FILE_FLAG_DELETE_ON_CLOSE | 关闭了上一次打开的句柄后，将文件删除。特别适合临时文件  |

● hTemplateFile：不为零，则指定一个文件句柄。新文件将从这个文件中复制扩展属性。

WritePrivateProfileString 函数向 INI 文件中写入指定节指定键名的字符串信息。原型如下：

```
BOOL WritePrivateProfileString(LPCTSTR lpAppName, LPCTSTR lpKeyName,
 LPCTSTR lpString, LPCTSTR lpFileName);
```

参数说明：

- lpAppName：即将写入的节名。
- lpKeyName：即将写入节名下的键名。
- lpString：即将写入节名下键名的数据值。
- lpFileName：INI 文件名，可以是全路径，如果不是全路径，默认在系统文件夹下新建一个 INI 文件。

## 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“提取指定文件夹目录到 INI 文件”。
- (2) 在窗体上添加两个按钮控件和一个列表框控件。
- (3) 创建 INI 文件，代码如下：

```
::GetCurrentDirectory(256,buf);//获取当前路径
strcat(buf,"\\list.ini");
HANDLE
handle=CreateFile(buf,GENERIC_READ|GENERIC_WRITE,FILE_SHARE_WRITE,NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_
NORMAL,NULL);//创建文件
if(handle!=NULL)
{
 CloseHandle(handle);
}
```

- (4) 向 INI 文件中写入数据，代码如下：

```
{
 // TODO: Add your control notification handler code here
 m_list.ResetContent();
 CFileFind file;
 if(Path.Right(1) != "\\")
 Path += "*.*";
 else
 Path += "*.*";
 BOOL bf;
 bf = file.FindFile(Path);//查找文件
 int i=1;
 while(bf)
 {
 bf = file.FindNextFile();//查找下一个文件
 if(!file.IsDots())
 {
 CString str;
 str.Format("%d",i);
```

```

strName = file.GetFileName();//获取文件名
m_list.AddString(strName);
::WritePrivateProfileString(_T(folder),_T(str),_T(strName),_T(buf));//向INI文件中写入数据
i++;
}
}
}

```

## 举一反三

根据本实例，读者可以：

- 提取文件夹目录信息到 Word 文档中。

## 5.3 与文件目录相关的命令操作

与文件目录相关的操作有删除文件目录、重命名文件目录等操作。本节将通过以下几个实例来说明如何在文件管理程序中运用与文件目录相关的命令。

## 实例 181 删除文件目录

本实例可以提高工作效率

实例位置：光盘\mingrisoft\05\181

## 实例说明

删除文件目录即删除文件夹，本实例主要实现删除文件夹的功能。运行程序，单击“浏览”按钮选择要删除的文件夹所在的路径，单击“删除”按钮，即可将该文件夹删除。实例运行结果如图 5.9 所示。

## 技术要点

利用 API 函数 RemoveDirectory 可以实现删除文件夹的功能。

RemoveDirectory 函数原型如下：

```

BOOL RemoveDirectory(LPCTSTR lpPathName);

```

参数 lpPathName 为要删除的文件路径。

说明：RemoveDirectory 函数只能删除空文件夹，因此在调用 RemoveDirectory 函数前需要删除目录下的文件。

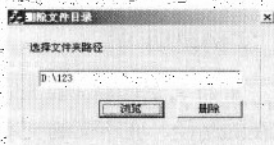


图 5.9 删除文件目录

## 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“删除文件目录”。
- (2) 向窗体中添加一个文本编辑框控件和一个按钮控件。
- (3) 主要程序代码：

```

void CDeleteFolderDlg::OnButDelete()
{
 // TODO: Add your control notification handler code here
 CString str;
 m_path.GetWindowText(str);
 DelFolder(str);
 //目录为空时删除目录
 if(RemoveDirectory(str))
 {
 MessageBox("删除成功!");
 return;
 }
}

//自定义函数，删除文件夹及其下文件
void CDeleteFolderDlg::DelFolder(CString path)
{
 CFileFind file;

```

```

if(path.Right(1) != "\\")
 path += "*.*";
BOOL bf;
bf = file.FindFile(path);
while(bf)
{
 bf = file.FindNextFile();
 //是文件时直接删除
 if (!file.IsDots() && !file.IsDirectory())
 DeleteFile(file.GetFilePath());
 else if (file.IsDots())
 continue;
 else if (file.IsDirectory())
 {
 path = file.GetFilePath();
 //是目录时, 继续递归调用函数删除该目录下的文件
 DelFolder(path);
 //目录为空后删除目录
 RemoveDirectory(path);
 }
}
}
}

```

### 举一反三

根据本实例, 读者可以:

- 删除指定目录下的系统日志文件。

## 实例 182 重命名文件目录

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\05\182

### 实例说明

重命名文件目录即修改文件夹的名称。运行程序, 单击“浏览”按钮选择要修改的文件夹所在的路径, 在下面的文本框中输入更改后的文件夹的名称, 单击“命名”按钮即完成文件夹的重命名操作。实例运行结果如图 5.10 所示。

### 技术要点

重命名文件目录其实就是创建一个新的文件夹, 然后把原文件夹中的文件复制到新文件夹中。本例使用 CreateDirectory 和 RemoveDirectory 函数创建和删除文件夹, 这两个函数前面已经介绍过了, 详细内容可参见前面实例。

### 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“重命名文件目录”。
- (2) 向窗体中添加两个文本编辑框控件和两个按钮控件。
- (3) 主要程序代码:

```

void CCallFolderDlg::OnButmingm()
{
 // TODO: Add your control notification handler code here
 CString str,strn;
 m_path.GetWindowText(str);
 m_name.GetWindowText(name);
 strn = str;
 strn.Replace(oName,name);//替换字符串
 CreateDirectory(strn,NULL);//创建目录
 DelFolder(str);
 RemoveDirectory(str);
}

void CCallFolderDlg::DelFolder(CString path)
{
}

```

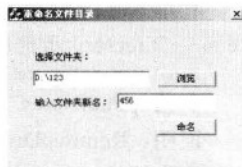


图 5.10 重命名文件目录

```

CFileFind file;
if(path.Right(1) != "\\")
{
 path += "*. *";
}
BOOL bf;
bf = file.FindFile(path);
while(bf)
{
 bf = file.FindNextFile();
 //是文件时直接删除
 if(!file.IsDots() && !file.IsDirectory())
 {
 CString path1;
 path1 = file.GetFilePath();
 path1.Replace(oName, name);
 //复制原文件到新文件夹中
 CopyFile(file.GetFilePath(), path1, true);
 DeleteFile(file.GetFilePath());
 }
 else if (file.IsDots())
 continue;
 else if (file.IsDirectory())
 {
 CString path2;
 path = file.GetFilePath();
 path2 = file.GetFilePath();
 //是目录时, 继续递归调用函数删除该目录下的文件
 path2.Replace(oName, name);
 //创建文件夹
 CreateDirectory(path2, NULL);
 DelFolder(path);
 //目录为空后删除目录
 RemoveDirectory(path);
 }
}
}
}

```

### 举一反三

根据本实例, 读者可以:

- 批量重命名文件夹。

## 5.4 文件、文件夹的复制和移动

本节将通过批量移动文件、网络文件夹的复制及在文件复制过程中显示进度条等实例系统地介绍文件、文件夹复制和移动方面的相关知识。

### 实例 183 批量移动文件

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\05\183

#### 实例说明

移动文件或批量移动文件在操作系统中是常见的操作, 通过应用程序也可以实现移动文件或批量移动文件的功能。运行程序, 单击“添加文件”按钮添加将要移动的文件, 然后单击“移动到”按钮将添加到列表中的文件移动到指定的文件夹中。程序运行如图 5.11 所示。

#### 技术要点

通过循环语句调用 MoveFile 方法实现批量移动文件的功能。下面介绍 MoveFile 方法。

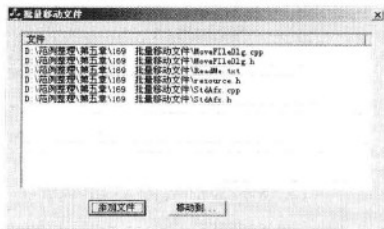


图 5.11 批量移动文件



MoveFile 方法用于实现在不同文件夹下移动文件, 相当于系统中的剪切命令, 语法如下:

```
BOOL MoveFile(LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName);
```

参数说明:

- lpExistingFileName: 指向源文件名的字符串指针。
- lpNewFileName: 指向目标文件名的字符串指针。

## 实现过程

(1) 新建对话框 MFC 工程。

(2) 在对话框上添加列表视图控件, ID 为 IDC\_FILELIST, 添加成员变量 m\_filelist, 添加两个按钮控件, ID 属性分别为 IDC\_BTADDFILE 和 IDC\_BTMOVE, Caption 属性分别为“添加文件”和“移动到...”。

(3) 主要程序代码:

```
BOOL CMoveFileDialog::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_filelist.SetExtendedStyle(LVS_EX_GRIDLINES); //修改列表视图样式
 m_filelist.InsertColumn(0, "文件", LVCFMT_LEFT, 450);
 return TRUE;
}
```

(4) “添加文件”按钮的实现函数, 该函数用于选择将要移动的文件, 代码如下:

```
void CMoveFileDialog::OnAddFile()
{
 CFileDialog log(TRUE, "文件", "*.*", OFN_HIDEREADONLY|
 OFN_ALLOWMULTISELECT, "FILE(*.*)*.*", NULL);
 if(log.DoModal()==IDOK)
 {
 //获取找开的所有文件, 添加到列表视图中
 POSITION pos = log.GetStartPosition();
 while(pos != NULL)
 {
 CString pathname= log.GetNextPathName(pos);
 m_filelist.InsertItem(m_filelist.GetItemCount(), pathname);
 }
 }
}
```

(5) “移动到...”按钮的实现函数, 该函数用于将列表中的文件移动到指定目录下, 代码如下:

```
void CMoveFileDialog::OnMove()
{
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer, MAX_PATH);
 bi.hwndOwner=GetSafeHwnd();
 bi.pidlRoot=NULL;
 bi.pszDisplayName=buffer;
 bi.lpszTitle="选择一个文件夹";
 bi.ulFlags=BIF_EDITBOX;
 bi.lpfm=NULL;
 bi.lParam=0;
 bi.ilImage=0;
 LPITEMIDLIST pList=NULL;
 if(pList=SHBrowseForFolder(&bi))!=NULL //显示文件浏览窗口
 {
 char path[MAX_PATH];
 ZeroMemory(path, MAX_PATH);
 SHGetPathFromIDList(pList, path);

 for(int i=0; i<m_filelist.GetItemCount(); i++)
 {
 CString pathtemp;
 pathtemp.Format("%s\\%s", path, GetNameFromPath(m_filelist.GetItemText(i, 0)));
 ::MoveFile(m_filelist.GetItemText(i, 0), pathtemp);
 }
 AfxMessageBox("移动文件完成");
 }
}
```

(6) 自定义函数实现, 从路径中得到文件名, 代码如下:

```

CString CMoveFileDialog::GetNameFromPath(CString path)
{
 CString strright;
 int pos=path.Find("\\");
 while(pos>0)
 {
 path=path.Right(path.GetLength()-1-pos);
 pos=path.Find("\\");
 }
 return path;
}

```

### 举一反三

根据本实例，读者可以：

- 清空文件夹中数据的同时备份数据。

## 实例 184 网络文件夹复制

本实例可以方便操作、提高效率

实例位置：光盘\mingrisoft\05\184

### 实例说明

本实例主要实现将网络文件夹中的内容复制到本地磁盘中。程序需要用户输入网络文件夹路径和目标文件夹路径，如图 5.12 所示，然后单击“复制文件”按钮，将文件复制到本地路径中。

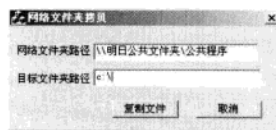


图 5.12 网络文件夹拷贝

### 技术要点

SHFileOperation 函数可以对任何数目的文件执行文件的复制、移动、更名及删除操作，包括通配符文件名和整个目录树，也包括网络文件名。其语法如下：

```
WINSHLAPI int WINAPI SHFileOperation(LPSHFILEOPSTRUCT lpFileOp);
```

参数说明：

- 4File OP：指定 SHFILEOPSTRUCT 结构。

SHFILEOPSTRUCT 结构的定义如下：

```

typedef struct _SHFILEOPSTRUCT{
 HWND hwnd;
 UINT wFunc;
 LPCSTR pFrom;
 LPCSTR pTo;
 FILEOP_FLAGS fFlags;
 BOOL fAnyOperationsAborted;
 LPVOID hNameMappings;
 LPCSTR lpszProgressTitle;
} SHFILEOPSTRUCT, FAR *LPSHFILEOPSTRUCT;

```

结构成员说明：

- hwnd：应用程序窗体句柄。
- wFunc：执行的操作，取值有 FO\_COPY、FO\_DELET、FO\_MOVE 和 FO\_RENAME。
- pFrom：源文件路径。
- pTo：目标文件路径。
- fFlags：控制文件操作的标识。
- fAnyOperationsAborted：操作完成前是否响应用户的取消操作。
- hNameMappings：名称映射句柄。
- lpszProgressTitle：进度条对话框名称。

注意：程序运行前请先登录到网络文件夹所在计算机。

## 实现过程

(1) 新建名为 NetworkFile 的对话框 MFC 工程。

(2) 在对话框上添加两个静态文本控件；添加两个文本编辑框控件，设置 ID 属性分别为 IDC\_EDNETWORK 和 IDC\_EDLOCAL；添加两个按钮控件，设置 ID 属性分别为 IDC\_BTCCOPY 和 IDC\_BTEXIT，设置 Caption 属性分别为“复制文件”和“取消”。

(3) 按钮“复制文件”的实现函数。该函数用于将网络中共享的文件夹中内容复制到本地文件夹下，代码如下：

```
void CNetworkFileDlg::OnCopy()
{
 CString strnetwork, strlocal;
 GetDlgItem(IDC_EDNETWORK)->GetWindowText(strnetwork);
 GetDlgItem(IDC_EDLOCAL)->GetWindowText(strlocal);
 if(strnetwork.IsEmpty())
 {
 AfxMessageBox("请输入网络文件夹路径");
 return;
 }
 if(strlocal.IsEmpty())
 {
 AfxMessageBox("请输入本地文件夹路径");
 return;
 }
 if(strnetwork.Left(2)!="\\")
 {
 AfxMessageBox("路径首部应是\\");
 return;
 }
 char fromname[80]="0";
 char toname[80]="0";

 strcpy(fromname, strnetwork);
 strcpy(toname, strlocal);
 strcat(fromname, "\\0");
 strcat(toname, "\\0");
 SHFILEOPSTRUCT lpFilestru;
 lpFilestru.hwnd=GetSafeHwnd();
 lpFilestru.wFunc=FO_COPY;
 lpFilestru.pFrom=fromname;
 lpFilestru.pTo=toname;
 lpFilestru.fFlags=FOF_ALLOWUNDO;
 lpFilestru.fAnyOperationsAborted=FALSE;
 BOOL bcopy=SHFileOperation(&lpFilestru);
 if(bcopy==0)
 {
 if(lpFilestru.fAnyOperationsAborted==TRUE)
 AfxMessageBox("复制被取消");
 else
 AfxMessageBox("复制成功");
 }
 else
 {
 AfxMessageBox("复制失败");
 }
}
```

(4) 按钮“取消”的实现函数。该函数用于关闭窗口体，代码如下：

```
void CNetworkFileDlg::OnExit()
{
 this->OnCancel();
}
```

## 举一反三

根据本实例，读者可以：

- 将系统数据备份到网络文件夹；
- 在局域网中传送文件；

● 通过局域网复制文件。

## 实例 185

## 文件复制过程中显示进度条

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\05\185

### 实例说明

等待复制文件是一个枯燥又漫长的过程，如果能在复制文件的过程中加上进度条，用户便可以随时看到文件复制的进度，就不会无限地等待了。本实例实现了在复制文件的过程中显示进度条的功能。通过单击两个“浏览”按钮分别实现选择将要复制的文件和选择复制文件所到的目标文件夹。单击“复制”按钮开始复制，复制的过程中，进度条将根据复制完成的比例来显示进度，如图 5.13 所示。

### 技术要点

实现本实例主要应用了 Progress 控件、CFile 类和 ldiv 函数。

Progress 控件首先要通过 SetRange 方法来设置进度条的数值范围，然后通过 SetPos 方法来设置显示的位置，SetPos 方法的设置值应该在 SetRange 方法所设的两个值之间。CFile 类主要完成文件的读取和写入，通过 ReadHuge 方法读取文件到缓存。ReadHuge 写入缓存数据到文件。ldiv 函数主要用于两个长整型数相除计算，可以协助计算文件复制完成的比例。

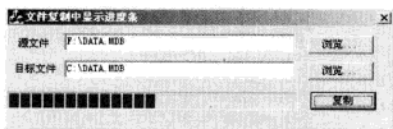


图 5.13 文件复制中显示进度条

### 实现过程

(1) 新建名为 FileCopy 的对话框 MFC 工程。

(2) 在对话框上添加 2 个静态文本控件；添加 2 个文本编辑框控件，设置 ID 属性分别为 IDC\_EDADD 和 IDC\_EDDEST；添加进度条控件，设置 ID 属性为 IDC\_FILEPROCESSER；添加成员变量 m\_fileproc；添加 3 个按钮控件，设置 ID 属性分别为 IDC\_ADD、IDC\_PUT 和 IDC\_COPY。

(3) 在头文件 FileCopyDlg.h 加入变量声明：

```
CString strname;
CString fullname;
CString pathname;
HGLOBAL hGlobal;
CFile* writefile;
CFile* readfile;
long readlen, poslen, filelen;
LPVOID pvData;
```

(4) 在 OnInitDialog 中初始化进度条，代码如下：

```
BOOL CFileCopyDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_fileproc.SetRange(0,100);
 m_fileproc.SetPos(0);
 return TRUE;
}
```

(5) 按钮控件 IDC\_ADD 的实现函数。该函数用于选择要复制的文件，代码如下：

```
void CFileCopyDlg::OnAdd()
{
 CFileDialog log(TRUE, "文件", "*.*", OFN_HIDEREADONLY, "FILE(*.*)*.*", NULL);
 if(log.DoModal() == IDOK)
 {
 pathname=log.GetPathName(); //获取路径
 strname=log.GetFileName(); //获取文件名
 GetDlgItem(IDC_EDADD)->SetWindowText(pathname);
 }
}
```



(6) 按钮控件 IDC\_PUT 的实现函数。该函数用于选择目标文件夹，代码如下：

```
void CFileCopyDlg::OnPut()
{
 if(strname.IsEmpty())
 return;
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer, MAX_PATH);
 bi.hwndOwner = GetSafeHwnd();
 bi.pidlRoot = NULL;
 bi.pszDisplayName = buffer;
 bi.lpszTitle = "选择一个文件夹";
 bi.ulFlags = BIF_EDITBOX;
 bi.lpfm = NULL;
 bi.lParam = 0;
 bi.iImage = 0;
 LPITEMIDLIST pList = NULL;
 if((pList = SHBrowseForFolder(&bi)) != NULL) //显示文件浏览窗口
 {
 char path[MAX_PATH];
 ZeroMemory(path, MAX_PATH);
 SHGetPathFromIDList(pList, path);
 fullname = path;
 if(fullname.Right(1) != "\\")
 fullname.Format("%s\\%s", path, strname);
 else
 fullname.Format("%s%s", path, strname);
 GetDlgItem(IDC_EDDEST)->SetWindowText(fullname);
 }
}
```

(7) 按钮控件 IDC\_COPY 的实现函数。该函数用于开始复制文件，代码如下：

```
void CFileCopyDlg::OnCopy()
{
 if(pathname.IsEmpty())
 return;
 if(fullname.IsEmpty())
 return;
 readfile = new CFile(pathname, CFile::modeRead);
 HANDLE
hfile = CreateFile(fullname, GENERIC_WRITE, GENERIC_WRITE, 0, 0, CREATE_NEW, FILE_ATTRIBUTE_NORMAL, 0);
 CloseHandle(hfile);
 writefile = new CFile(fullname, CFile::modeWrite);
 filelen = readfile->GetLength(); //获取文件大小
 //计算文件的百分之一大小
 ldiv_t r;
 r = ldiv(filelen, 100);
 long pos = r.quot;
 //保存递增的百分比大小
 long ipos;
 ipos = pos;
 int i = 0;
 hGlobal = GlobalAlloc(GMEM_MOVEABLE, 512);
 pvData = GlobalLock(hGlobal);
 while(1) {
 ZeroMemory(pvData, 512);
 readlen = readfile->ReadHuge(pvData, 512);
 //获得文件指针的位置
 poslen = readfile->GetPosition();
 //判断滚动条是否可以增加进度
 if(poslen > ipos)
 {
 ipos += pos;
 i++;
 }
 m_fileproc.SetPos(i);
 m_fileproc.Invalidate();
 writefile->WriteHuge(pvData, readlen);
 //如果文件指针移到末尾就退出循环
 if(poslen == filelen)
 break;
 }
 AfxMessageBox("复制完成");
 m_fileproc.SetPos(0);
}
```

```
GlobalUnlock(hGlobal);
readfile->Close();
writefile->Close();
}
```

### 举一反三

根据本实例，读者可以：

- 下载文件时显示进度条；
- 打开文件时显示进度条。

## 5.5 文件修改

本节主要介绍如何更改文件夹图标、批量修改和批量删除文件等相关操作，通过这几个方面的内容使读者更加深刻地了解与文件相关的编程知识。

### 实例 186 修改应用程序图标

这是一个自娱自乐的实例

实例位置：光盘\mingrisoft\05\186

### 实例说明

网上的许多软件能够修改可执行文件的图标，如何实现呢？其实，用户只要了解 PE 档案格式，修改应用程序图标也就容易了。本例便是根据 PE 档案格式修改可应用程序的图标，如图 5.14 所示。

### 技术要点

若想修改应用程序的图标，必须修改其可执行文件的内部信息，也就是对应用程序的 PE 档案格式中的数据进行修改。

Portable Executable 简称 PE，是 Windows 应用程序的档案格式。了解 PE 格式，能够使用户熟悉应用程序的组成。对于一些破坏应用程序的病毒，杀毒软件主要是通过查看应用程序的 PE 档案了解其是否被感染病毒。

PE 档案主要由 DOS 头、PE 表头、段等几部分组成，如图 5.15 所示。

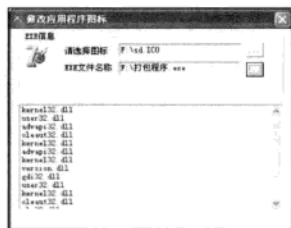


图 5.14 修改应用程序图标

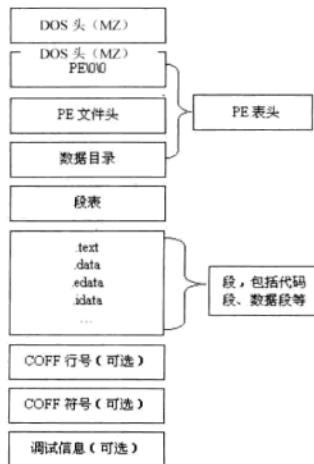


图 5.15 PE 档案格式

对于 DOS 头，对应的结构为 IMAGE\_DOS\_HEADER，其中，e\_lfanew 成员表示 DOS 头的开始位置相对于 PE 表头的偏移量。PE 表头由 IMAGE\_NT\_HEADERS 结构表示，包含有 PE 的签名、PE 文件头、数据目录等信息。在 PE 表头之后是段表，对应的结构为 IMAGE\_SECTION\_HEADER，其中包含有段名称、段原始数据的偏移量等信息。在程序中，可以通过段表获得段的实际位置。

为了修改可执行文件的图标，需要获取图标在可执行文件中的位置。通常，图标数据存储在应用程序的三级资源目录下。用户首先获得的资源目录是一级资源目录，可以通过一级资源目录获得一级资源目录实体；然后通过目录实体获得二级资源目录，依此类推，可以获得三级资源目录和三级资源目录实体；最后通过三级资源目录实体获得图标的实际数据。在修改图标数据时，需要注意的是从某个图标文件中加载的图

标数据包含有图标目录、图标索引等信息,而可执行文件中的图标数据不包含这些信息,它存储的是实际的图标数据。为了将某个图标文件写入到可执行文件中的图标数据,需要将加载的图标文件偏移 22 个字节。因为前 22 个字节表示的是图标的文件格式。

## 实现过程

(1) 新建名为 FetchDll 的对话框 MFC 工程。

(2) 在对话框上添加两个静态文本控件;添加两个文本编辑框控件;添加两个图片控件,将 Type 属性都设置为 Icon;添加两个按钮控件;添加一个列表框控件。

(3) 主要程序代码。

```
void CFetchDllDlg::OnOK()
{
 UpdateData();
 if (m_FileName.IsEmpty())
 {
 MessageBox("请选择可执行文件");
 return;
 }
 m_List.ResetContent();

 HANDLE hFile; //文件句柄
 HANDLE hMap; //内存映射句柄
 char *pBase; //PE文件基地址
 IMAGE_DOS_HEADER* peDos_Header; //DOS头
 IMAGE_NT_HEADERS* peNT_Header; //PE文件头
 IMAGE_SECTION_HEADER* peSection_Header; //段表头
 IMAGE_IMPORT_DESCRIPTOR* peImport_Descript; //引用表
 IMAGE_RESOURCE_DIRECTORY* peResource_Dir;
 IMAGE_RESOURCE_DIRECTORY_ENTRY* peResource_Entry;

 //创建可读写文件
 hFile=CreateFile(m_FileName.GetBuffer(0),GENERIC_READ|
 GENERIC_WRITE,FILE_SHARE_READ,0,OPEN_EXISTING,
 FILE_ATTRIBUTE_NORMAL,0);
 if (hFile==INVALID_HANDLE_VALUE)
 {
 return;
 }
 //创建文件映象
 if (!hMap=CreateFileMapping(hFile,0,PAGE_READWRITE|SEC_COMMIT,0,0,0))
 {
 CloseHandle(hFile);
 CloseHandle(hMap);
 return;
 }
 if (!(pBase=(char*)::MapViewOfFile(hMap,FILE_MAP_ALL_ACCESS|
 FILE_MAP_COPY,0,0,0)))//创建文件映象视图
 {
 CloseHandle(hFile);
 CloseHandle(hMap);
 return;
 }

 peDos_Header=(IMAGE_DOS_HEADER *)pBase; //获取PE文件格式的起始地址

 //e_lfanew是相对虚地址,即偏移量,指向真正的PE表头
 peNT_Header=(IMAGE_NT_HEADERS *) (pBase+peDos_Header->e_lfanew); //NT头指针地址

 //获取段表的数量
 int secCount = peNT_Header->FileHeader.NumberOfSections;

 //获取段表头的位置
 peSection_Header = (IMAGE_SECTION_HEADER *) ((char*)
 peNT_Header+sizeof(IMAGE_NT_HEADERS));
 char* pDllName = NULL;

 //宏Addr根据某个段中的相对地址获取实际地址
 #define Addr(offset) (void*) ((char*) (char*) pBase+
 peSection_Header->PointerToRawData) + ((DWORD)(offset)-
 peSection_Header->VirtualAddress))

 #define isValid(addr,begin,len) ((char*)(addr)>=(char*)(begin) &&
 (char*)(addr)<=(char*)(begin)+(len))
```

```

//遍历段表
for (int i = 0; i < secCount; i++, peSection_Header++)
{
 for(int directory = 0; directory <
IMAGE_NUMBEROF_DIRECTORY_ENTRIES; directory++)
 {
 if(peNT_Header->OptionalHeader.DataDirectory[directory].VirtualAddress
&& isValid(peNT_Header->OptionalHeader.DataDirectory[directory]
.VirtualAddress, peSection_Header->VirtualAddress,
peSection_Header->SizeOfRawData))
 { //引入表
 if (directory == IMAGE_DIRECTORY_ENTRY_IMPORT)
 {
 //获取引入表在.idata段中的位置
 peImport_Descript = (IMAGE_IMPORT_DESCRIPTOR *)
Addr(peNT_Header->OptionalHeader.DataDirectory[directory].VirtualAddress);
 for (int j =
0; !IsBadReadPtr(peImport_Descript, sizeof(*peImport_Descript)) &&
peImport_Descript->Name; peImport_Descript++, j++)
 {
 pDllName = (char*)Addr(peImport_Descript->Name);
 m_List.AddString(pDllName);
 }
 }
 else if (directory == IMAGE_DIRECTORY_ENTRY_RESOURCE)
 {
 //一级资源目录
 peResource_Dir =
(IMAGE_RESOURCE_DIRECTORY*)Addr(peNT_Header->OptionalHeader.DataDirectory[directory].VirtualAddress);

 //资源数量
 int ReCount = peResource_Dir->NumberOfIdEntries + peResource_Dir->NumberOfNamedEntries;

 //一级资源目录实体入口
 peResource_Entry =
(IMAGE_RESOURCE_DIRECTORY_ENTRY*)((char*)peResource_Dir + sizeof(IMAGE_RESOURCE_DIRECTORY));

 IMAGE_RESOURCE_DIRECTORY* peTempDir, *pe3Dir, *pe4Dir;

 IMAGE_RESOURCE_DIRECTORY_ENTRY* peTempEntry, *peIconDir;
 IMAGE_RESOURCE_DATA_ENTRY* peDataEntry;

 for (int m = 0; m < ReCount; m++, peResource_Entry++)
 {
 int type = peResource_Entry->Id;
 //3表示图标, 5表示对话框, 14表示图标组

 if (type == 3)
 {
 //进入二级目录
 peTempDir =
(IMAGE_RESOURCE_DIRECTORY*)((char*)peResource_Dir + peResource_Entry->OffsetToDirectory);

 //获取二级目录实体数量
 int iconCount = peTempDir->NumberOfIdEntries + peTempDir->NumberOfNamedEntries;

 //二级目录实体入口
 peTempEntry =
(IMAGE_RESOURCE_DIRECTORY_ENTRY*)((char*)peTempDir + sizeof(IMAGE_RESOURCE_DIRECTORY));

 for (int c = 0; c < iconCount; c++, peTempEntry++)
 {
 if (peTempEntry->DataIsDirectory > 0)
 {
 //三级目录
 pe3Dir =
(IMAGE_RESOURCE_DIRECTORY*)((char*)peResource_Dir + peTempEntry->OffsetToDirectory);
 //三级实体
 peIconDir = (IMAGE_RESOURCE_DIRECTORY_ENTRY*)((char*)pe3Dir +
sizeof(IMAGE_RESOURCE_DIRECTORY));

 //图标ID
 int id = peTempEntry->Id;
 peDataEntry = (IMAGE_RESOURCE_DATA_ENTRY*)
((char*)peResource_Dir + peIconDir->OffsetToDirectory);

```



```

 int size = peDataEntry->Size;
 unsigned char* pData;
 if (c==0)
 {
 pData = (unsigned char*)((char*) Addr(peDataEntry->OffsetToData));

 if (!m_IconFile.IsEmpty())
 {
 CFile file;
 int len = file.GetLength();
 m_plconData = new char[len+1];
 memset(m_plconData, 0, len+1);

 file.ReadHuge(m_plconData, len);

 char* temp = m_plconData;
 m_plconData += 22;
 file.Close();

 memcpy(pData, m_plconData, size);

 delete temp;
 }
 }
 }
}

FlushFileBuffers(hFile);
CloseHandle(hMap);
CloseHandle(hFile);
SetFileAttributes(m_FileName, FILE_ATTRIBUTE_READONLY);
SetFileAttributes(m_FileName, FILE_ATTRIBUTE_NORMAL);
}

```

### 举一反三

根据本实例，读者可以：

- 提取可执行文件图标；
- 将应用程序图标修改成另一个应用程序图标。

## 实例 187 更改文件夹图标

本实例可以提高工作效率

实例位置：光盘\mingrisoft\05\187

### 实例说明

本实例实现如何更改系统默认的文件夹图标。运行程序，如图 5.16 所示，单击“浏览”按钮选择将要修改图标的文件夹，单击“选择图标”按钮添加一个图标文件，单击“修改图标”按钮实现修改目标文件夹的图标。

### 技术要点

文件夹图标的修改需要在目标文件夹下新建一个 Desktop.ini 文件，并在 INI 文件中添加键值。具体过程为在 INI 文件的[ShellClassInfo]节下，添

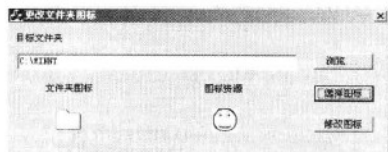


图 5.16 更改文件夹图标

加 IconFile=图标 1.ico 和 IconIndex=0。IconFile 是指定图标文件，最好和 INI 文件在同一文件夹下，如果不在同一文件夹下，应使用图标资源的全路径；IconIndex 是指定图标的索引，一般为零。本实例通过多个 API 函数来实现这一过程，具体如下。

- (1) 使用 SHBrowseForFolder 函数实现打开文件夹选择对话框。
- (2) 使用 SHGetFileInfo 函数实现获取文件夹图标信息。
- (3) 使用 SetFileAttributes 函数实现设置文件属性。
- (4) 使用 CopyFile 函数实现复制文件。
- (5) 使用 WritePrivateProfileString 函数实现向 INI 文件中写入键值。

 注意 ReadIconFromICOFile 是 CIcons 对象的方法。

## 实现过程

- (1) 新建名为 ChangeDirIcon 的对话框 MFC 工程。

(2) 在对话框上添加 3 个静态文本控件；添加文本编辑控件，设置 ID 属性为 IDC\_EDSELDIR；添加 2 个图片控件，设置 ID 属性分别为 IDC\_DIR 和 IDC\_DES，将 Type 属性都设置为 Icon；添加 3 个按钮控件，设置 ID 属性分别为 IDC\_BTADD、IDC\_BTSEL 和 IDC\_BTMODIFY，Caption 属性分别为“浏览...”、“选择图片”和“修改图标”。

- (3) 在工程中添加 4 个文件，分别是 Dib.h、Dib.CPP、Icon.h 和 Icon.CPP。

- (4) 在 ChangeDirIconDlg.h 文件中加入变量声明和头文件引用：

```
#include "Icons.h"
CString strpath,extname,despath,desname;
LPICONRESOURCE lpIR;
CIcons *pIcons;
```

- (5) “浏览”按钮的实现函数。该函数用于选择要修改图标的文件夹，代码如下：

```
void CChangeDirIconDlg::OnAdd()
{
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer,MAX_PATH);
 bi.hwndOwner=GetSafeHwnd();
 bi.pidlRoot=NULL;
 bi.pszDisplayName=buffer;
 bi.lpszTitle="选择一个文件夹";
 bi.ulFlags=BIF_EDITBOX;
 bi.lpfm=NULL;
 bi.lParam=0;
 bi.ilImage=0;
 LPITEMIDLIST pList=NULL;
 if(pList=SHBrowseForFolder(&bi))!=NULL) //显示文件浏览窗口
 {
 char path[MAX_PATH];
 ZeroMemory(path,MAX_PATH);
 SHGetPathFromIDList(pList,path);
 GetDlgItem(IDC_EDSELDIR)->SetWindowText(path);
 SHFILEINFO shfile;
 despath=path;
 strcat(path,"\\");
 ::SHGetFileInfo(path,0,&shfile,sizeof(shfile),SHGFI_ICON);
 CStatic*pStatic = (CStatic*)GetDlgItem(IDC_DIR);
 pStatic->SetIcon(shfile.hIcon);
 }
}
```

- (6) 按钮“选择图标”实现函数。该函数用于添加图标文件，代码如下：

```
void CChangeDirIconDlg::OnSel()
{
 CFileDialog fileDialog(TRUE, "*.ICO", NULL, NULL,
 "资源文件(*.ICO;*.BMP;*.EXE;*.DLL;*.ICL)|*.ICO;*.BMP;*.EXE;*.DLL;*.ICL|");
 if(fileDialog.DoModal()==IDOK)
 {
 strpath=fileDialog.GetPathName(); //获取路径
 desname=fileDialog.GetFileName(); //获取文件名
 }
}
```

```

 extname= fileDialog.GetFileExt(); //获取文件扩展名
 extname.MakeLower();
 }
 if(extname=="ico")
 {
 lpIR=plcons->ReadIconFromICOFile(strpath);
 HICON hlcon;
 hlcon=ExtractIcon(AfxGetInstanceHandle(),strpath,0); //获取图标
 CStatic* pStatic = (CStatic*)GetDlgItem(IDC_DES);
 pStatic->SetIcon(hlcon);
 }
}

```

(7) “修改图标”按钮的实现函数。该函数用于对文件夹的图标进行修改,代码如下:

```

void CChangeDirIconDlg::OnModify()
{
 CString temp,temp2;
 SetFileAttributes(despath,FILE_ATTRIBUTE_READONLY); //设置文件属性
 temp=despath;
 temp+="\\";
 temp+=desname;
 if(CopyFile(strpath,temp,FALSE))//文件复制
 {AfxMessageBox("成功");}
 else
 {AfxMessageBox("失败");}
 temp2=despath;
 temp2+="\\";
 temp2+="desktop.ini";
 ::WritePrivateProfileString("ShellClassInfo","IconFile",desname,temp2);
 ::WritePrivateProfileString("ShellClassInfo","IconIndex","0",temp2);
}

```

### 举一反三

根据本实例,读者可以:

- 对文件夹进行管理;
- 批量更改文件夹的图标。

## 实例 188 批量删除指定类型的文件

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\05\188

### 实例说明

对于系统中的垃圾文件,可以进行批量删除,例如批量删除某一类型的文件。运行程序,单击“添加文件”按钮将要批量删除的文件加入到列表中,在向列表添加文件的过程中可以通过扩展名进行特定文件类型的过滤,如图 5.17 所示,单击“删除”按钮,将批量删除列表中的文件。

### 技术要点

通过循环语句调用 DeleteFile 方法实现批量删除。DeleteFile 方法实现删除磁盘文件,语法如下:

```
BOOL DeleteFile(LPCTSTR lpFileName);
```

参数说明:

- lpFileName: 指向将要删除文件名字符串指针。

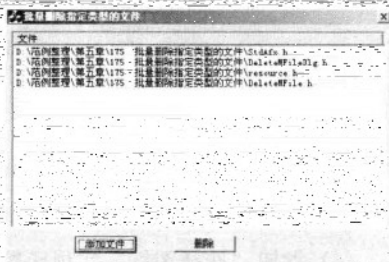


图 5.17 批量删除指定类型的文件

## 实现过程

(1) 新建名为 DeleteMFile 的对话框 MFC 工程。

(2) 在对话框上添加列表视图控件, 设置 ID 属性为 IDC\_FILELIST, 添加成员变量 m\_filelist; 添加两个按钮控件, 设置 ID 属性分别为 IDC\_BTADDFILE 和 IDC\_BTDEL, Caption 属性分别为“添加文件”和“删除”。

(3) 在 OnInitDialog 中进行列表控件的初始化, 代码如下:

```
BOOL CDeleteMFileDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_filelist.SetExtendedStyle(LVS_EX_GRIDLINES);
 m_filelist.InsertColumn(0, "文件", LVCFMT_LEFT, 450);
 return TRUE;
}
```

(4) “添加文件”按钮的实现函数。该函数用于添加将要删除的文件, 代码如下:

```
void CDeleteMFileDlg::OnAdd()
{
 CFileDialog log(TRUE, "文件", "*", OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT|
 OFN_ALLOWMULTISELECT, "FILE(*.*)*.jpeg(*.jpg)*.jpg|
 文本(*.txt)*.txt|", NULL);
 if(log.DoModal() == IDOK) //显示文件打开对话框
 {
 POSITION pos = log.GetStartPosition();
 while(pos != NULL)
 {
 CString pathname=log.GetNextPathName(pos);
 m_filelist.InsertItem(m_filelist.GetItemCount(), pathname);
 }
 }
}
```

(5) “删除”按钮的实现函数。该函数用于删除列表控件中的文件, 代码如下:

```
void CDeleteMFileDlg::OnDelete()
{
 int count=m_filelist.GetItemCount(); //获取文件数量
 for(int i=0; i<count; i++)
 {
 ::DeleteFile(m_filelist.GetItemText(i, 0)); //删除文件
 }
 AfxMessageBox("删除成功");
 m_filelist.DeleteAllItems();
}
```

## 举一反三

根据本实例, 读者可以:

- 清除应用程序日志;
- 批量删除系统中过时无用的数据。

## 实例 189 批量重命名文件

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\05\189

## 实例说明

本实例完成文件的批量命名。单击“添加文件”按钮, 将要修改文件名的文件添加到列表, 然后在文本框输入一个文件名, 单击“重命名为”按钮后, 程序将以输入的字符后面加数字的方式对列表中的文件进行批量重命名。程序运行如图 5.18 所示, 修改文件名后的结果如图 5.19 所示。



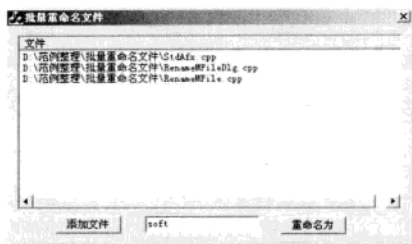


图 5.18 批量重命名文件

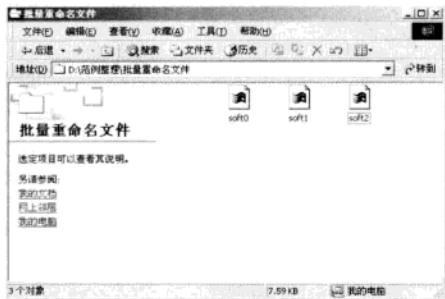


图 5.19 修改结果

## 技术要点

本实例主要通过循环语句调用 rename 函数来实现批量重命名文件的功能。rename 是 stdio.h 中定义的函数，它的语法如下：

```
int rename(const char *oldname, const char *newname);
```

参数说明：

- oldname：指向修改前文件名字符串指针。
- newname：指向修改后文件名字符串指针。
- 返回值：如果函数执行成功，则返回零。

注意：向列表添加文件应该是统一扩展名的文件。

## 实现过程

(1) 新建名为 RenameMFile 的对话框 MFC 工程。

(2) 在对话框上添加列表视图控件，设置 ID 属性为 IDC\_FILELIST，添加成员变量 m\_filelist；添加文本编辑框控件，设置 ID 属性为 IDC\_EDNAME；添加两个按钮控件，设置 ID 属性分别为 IDC\_BTADDFILE 和 IDC\_BTRENAME，Caption 属性分别为“添加文件”和“重命名为”。

(3) 主要程序代码：

```
BOOL CRenameMFileDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 //此处代码省略
 m_filelist.SetExtendedStyle(LVS_EX_GRIDLINES);
 m_filelist.InsertColumn(0, "文件", LVCFMT_LEFT, 450);
 return TRUE;
}
```

(4) “添加文件”按钮的实现函数。该函数用于添加将要重命名的文件，代码如下：

```
void CRenameMFileDlg::OnAddFile()
{
 CFileDialog log(TRUE, "文件", "*.*", OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT|
 OFN_ALLOWMULTISELECT, "FILE(*.*)"*. *jpeg(*.jpg)*.jpg|
 文本(*.txt)*.txt|", NULL);
 if(log.DoModal() == IDOK) //显示文件打开对话框
 {
 POSITION pos = log.GetStartPosition();
 while(pos != NULL)
 {
 CString pathname=log.GetNextPathName(pos);
 m_filelist.InsertItem(m_filelist.GetItemCount(), pathname);
 }
 }
}
```

(5) “重命名为”按钮的实现函数。该函数用于对列表控件中的文件进行重命名，代码如下：

```
void CRenameMFileDlg::OnReName()
{
 CString strname, strtemp, nametemp, strext;
 GetDlgItem(IDC_EDNAME)->GetWindowText(strname);
 if(strname.IsEmpty())return;
}
```

```

int count = m_filelist.GetItemCount();//获取文件数量
for(int i=0;i<count;i++)
{
 nametemp=m_filelist.GetItemText(i,0);
 strtemp=m_filelist.GetItemText(i,0);
 if(strtemp.Right(4).GetAt(0)=='.')
 strext=strtemp.Right(3);
 else
 {
 if(strtemp.Right(2).GetAt(0)=='')
 strext=strtemp.Right(1);
 else
 strext="";
 }
 int pos=nametemp.Find("\\");
 while(pos>0)
 {
 nametemp=nametemp.Right(nametemp.GetLength()-1-pos);
 pos=nametemp.Find("\\");
 }
 strtemp=strtemp.Left(strtemp.GetLength()-nametemp.GetLength());
 CString temp;temp.Format("%s%s%d.%s",strtemp,strname,i,strext);
 ::rename(nametemp,temp);//文件重命名
}
AfxMessageBox("批量重命名成功");
}

```

### 举一反三

根据本实例，读者可以：

- 批量重命名某一类文件；
- 批量重命名书籍稿件。

## 实例 190 修改文件属性

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\05\190

### 实例说明

在 Windows 操作系统当中，可以通过文件的属性对话框获取文件的属性信息，本实例也实现这样的功能，即获取文件大小和时间等属性，并可修改文件只读、隐藏和系统等方面的属性。运行程序单击“添加文件”按钮，打开指定文件，并获得文件属性，如图 5.20 所示，单击“设置”按钮可以将复选框中指定属性设置给文件，但目录和卷标属于特殊属性，一般不能对文件进行设置。

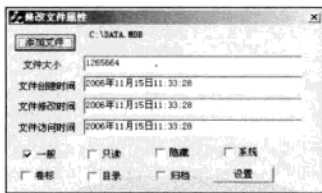


图 5.20 修改文件属性

### 技术要点

本实例主要通过 CFile 类的 GetStatus 方法和 SetStatus 方法实现，GetStatus 方法是获得文件属性，语法如下：

```

BOOL GetStatus(CFileStatus& rStatus) const;
static BOOL PASCAL GetStatus(LPCTSTR lpszFileName, CFileStatus& rStatus);

```

参数说明：

- rStatus：指向 CFileStatus 结构的对象，CFileStatus 的结构成员如表 5.4 所示。

表 5.4 CFileStatus 的结构成员

| 结构成员          | 描述     |
|---------------|--------|
| CTime m_ctime | 文件创建时间 |
| CTime m_mtime | 文件修改时间 |
| CTime m_atime | 文件访问时间 |

续表

| 结 构 成 员                      | 描 述  |
|------------------------------|------|
| LONG m_size                  | 文件大小 |
| BYTE m_attribute             | 文件属性 |
| char m_szFullName[_MAX_PATH] | 文件名  |

- lpzFileName: 指向文件名的字符串指针。

SetStatus 方法是设置文件属性, 语法如下:

```
static void SetStatus(LPCTSTR lpzFileName, const CFileStatus& status);
```

参数说明:

- lpzFileName: 指向文件名的字符串指针。
- rStatus: 指向 CFileStatus 结构的对象。

## 实现过程

(1) 新建名为 SetFileAttri 的对话框 MFC 工程。

(2) 在对话框上添加 5 个静态文本控件, 设置其中一个 ID 属性为 IDC\_FILENAME, Caption 属性为空, 添加成员变量 m\_filename; 添加 4 个文本编辑框控件, 设置 ID 属性分别为 IDC\_EDFILESIZE、IDC\_EDCREATETIME、IDC\_EDMODIFYTIME 和 IDC\_EDREADTIME; 添加 7 个复选按钮控件, 并添加成员变量 m\_vol、m\_system、m\_readonly、m\_normal、m\_hide、m\_dir 和 m\_ar, 添加 2 个按钮控件, 设置 ID 属性分别为 IDC\_BTADD 和 IDC\_BTSET, Caption 属性分别为“添加文件”和“设置”。

(3) “添加文件”按钮的实现函数。该函数用于添加需要修改文件属性的文件, 代码如下:

```
void CSetFileAttriDlg::OnAdd()
{
 CFileDialog log(TRUE, "文件", "*", "*", OFN_HIDEREADONLY, "FILE(*.*)", NULL);
 if(log.DoModal() == IDOK)
 {
 pathname=log.GetPathName();//获取路径
 m_filename.SetWindowText(pathname);
 CFileStatus status;
 CFile::GetStatus(pathname, status);//获取文件状态
 CString strsize;
 strsize.Format("%d", status.m_size);
 CTime createtime=status.m_ctime;
 CTime modifytime=status.m_mtime;
 CTime readtime=status.m_atime;
 GetDlgItem(IDC_EDFILESIZE)->SetWindowText(strsize);
 GetDlgItem(IDC_EDCREATETIME)->SetWindowText(
 createtime.Format("%Y年%m月%d日%H:%M:%S"));
 GetDlgItem(IDC_EDMODIFYTIME)->SetWindowText(
 modifytime.Format("%Y年%m月%d日%H:%M:%S"));
 GetDlgItem(IDC_EDREADTIME)->SetWindowText(
 readtime.Format("%Y年%m月%d日%H:%M:%S"));
 if(status.m_attribute & 0x00)
 m_ar.SetCheck(1);
 if(status.m_attribute & 0x10)
 m_dir.SetCheck(1);
 if(status.m_attribute & 0x02)
 m_hide.SetCheck(1);
 if(status.m_attribute & 0x20)
 m_normal.SetCheck(1);
 if(status.m_attribute & 0x01)
 m_readonly.SetCheck(1);
 if(status.m_attribute & 0x04)
 m_system.SetCheck(1);
 if(status.m_attribute & 0x08)
 m_vol.SetCheck(1);
 }
}
```

(4) “设置”按钮的实现函数。该函数用于设置文件指定的属性, 代码如下:

```

void CSetFileAttrDlg::OnSet()
{
 CFileStatus status;
 BYTE m_newattri;
 if(m_ar.GetCheck())
 {
 m_newattri=0x20;
 CFile::GetStatus(pathname,status);//获取文件状态
 status.m_attribute=m_newattri;
 CFile::SetStatus(pathname,status);//设置文件状态
 }
 if(m_dir.GetCheck())
 {
 m_newattri=0x10;
 CFile::GetStatus(pathname,status);
 status.m_attribute=m_newattri;
 CFile::SetStatus(pathname,status);
 }
 if(m_hide.GetCheck())
 {
 m_newattri=0x02;
 CFile::GetStatus(pathname,status);
 status.m_attribute=m_newattri;
 CFile::SetStatus(pathname,status);
 }
 if(m_normal.GetCheck())
 {
 m_newattri=0x00;
 CFile::GetStatus(pathname,status);
 status.m_attribute=m_newattri;
 CFile::SetStatus(pathname,status);
 }
 if(m_readonly.GetCheck())
 {
 m_newattri=0x00;
 CFile::GetStatus(pathname,status);
 status.m_attribute=m_newattri;
 CFile::SetStatus(pathname,status);
 m_newattri=0x01;
 status.m_attribute=m_newattri;
 CFile::SetStatus(pathname,status);
 }
 if(m_system.GetCheck())
 {
 m_newattri=0x04;
 CFile::GetStatus(pathname,status);
 status.m_attribute=m_newattri;
 CFile::SetStatus(pathname,status);
 }
 if(m_vol.GetCheck())
 {
 m_newattri=0x08;
 CFile::GetStatus(pathname,status);
 status.m_attribute=m_newattri;
 CFile::SetStatus(pathname,status);
 }
}

```

### 举一反三

根据本实例，读者可以：

- 自动获取文件的名称。

## 实例 191 修改文件及目录的名称

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\05\191

### 实例说明

本实例实现了修改文件和目录的名称的功能。修改文件名称和修改目录名称是两个独立的



部分。修改目录名称要先选择要修改目录名称的文件夹，然后在编辑框中输入新的名称，通过“修改”按钮就可以实现名称的修改；修改文件名称也要先选择文件，然后在编辑框中输入新的名称，通过“修改”按钮名称的修改，如图 5.21 所示。

## 技术要点

本实例主要通过 MoveFile 函数来修改目录的名称，通过 CFile 类的 Rename 方法来修改文件名称。MoveFile 函数主要用来移动文件或目录，通过它可以达到修改目录名称的目的，其语法如下：

```
BOOL MoveFile(LPCTSTR lpExistingFileName,LPCTSTR lpNewFileName);
```

参数说明：

- lpExistingFileName：指向修改前文件名的字符串指针。
- lpNewFileName：指向修改后文件名的字符串指针。

CFile 类的 Rename 方法主要实现对文件名字的修改，语法如下：

```
static void PASCAL Rename(LPCTSTR lpszOldName, LPCTSTR lpszNewName);
```

参数说明：

- lpszOldName：指向修改前文件名的字符串指针。
- lpszNewName：指向修改后文件名的字符串指针。

## 实现过程

(1) 新建名为 RenameFile 的对话框 MFC 工程。

(2) 在对话框上添加 4 个静态文本控件；添加 4 个文本编辑框控件，设置 ID 属性分别为 IDC\_EDPATHNAME、IDC\_EDPATHNAMESET、IDC\_EDNAME 和 IDC\_EDNAMESET；添加 4 个按钮控件，设置 ID 属性分别为 IDC\_BTPATHADD、IDC\_BTPATHSET、IDC\_BTNAMEADD 和 IDC\_BTNAMESET，Caption 属性分别为“...”、“修改”、“...”和“修改”。

(3) 在头文件 RenameFileDlg.h 中加入变量声明：

```
CString strpathname;
CString strfilename;
CString pathofpath;
```

(4) “...”（目录修改）按钮的实现函数。该函数用于选择要修改名称的目录，代码如下：

```
void CRenameFileDlg::OnPathAdd()
{
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer,MAX_PATH);
 bi.hwndOwner=GetSafeHwnd();
 bi.pidlRoot=NULL;
 bi.pszDisplayName=buffer;
 bi.lpszTitle="选择一个文件夹";
 bi.ulFlags=BIF_EDITBOX;
 bi.lpfm=NULL;
 bi.lParam=0;
 bi.ilImage=0;
 LPITEMIDLIST pList=NULL;
 if(pList=SHBrowseForFolder(&bi))!=NULL //显示文件浏览窗口
 {
 char path[MAX_PATH];
 ZeroMemory(path,MAX_PATH);
 SHGetPathFromIDList(pList,path);
 pathofpath=strpathname+path;
 CString pathname,temp;
 pathname=path;
 int pos=pathname.Find("\\");
 while(pos>0)
 {
```

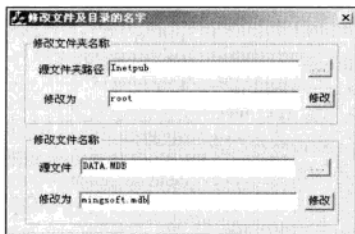


图 5.21 修改文件及目录的名称

```

 pathname=pathname.Right(pathname.GetLength()-1-pos);
 pos=pathname.Find("\\");
 }
 GetDlgItem(IDC_EDPATHNAME)->SetWindowText(pathname);
 pathofpath=pathofpath.Left (pathofpath. GetLength() -pathname.GetLength());
}

```

(5) “...” (文件修改) 按钮的实现函数。该函数用于选择要修改名称的文件, 代码如下:

```

void CRenameFileDialog::OnNameAdd()
{
 CFileDialog log(TRUE, "文件", "*.*, OFN_HIDEREADONLY|
 OFN_ALLOWMULTISELECT, "FILE(*.*)*.*", NULL);
 if(log.DoModal()==IDOK)//显示文件打开对话框
 {
 strfilename=log.GetPathName();
 CString filename=log.GetFileName();
 GetDlgItem(IDC_EDNAME)->SetWindowText(filename);
 }
}

```

(6) “修改 (目录)” 按钮的实现函数。该函数用于修改目录名称, 代码如下:

```

void CRenameFileDialog::OnPathSet()
{
 if(strpathname.IsEmpty())
 return;
 CString edpathname;
 GetDlgItem(IDC_EDPATHNAMESET)->GetWindowText(edpathname);
 if(edpathname.IsEmpty())
 return;
 pathofpath+=edpathname;
 if(::MoveFile(strpathname,pathofpath))
 AfxMessageBox("修改成功");
}

```

(7) “修改 (文件)” 按钮的实现函数。该函数用于修改文件名称, 代码如下:

```

void CRenameFileDialog::OnNameSet()
{
 if(strfilename.IsEmpty())
 return;
 CString stredfilename;
 GetDlgItem(IDC_EDNAMESET)->GetWindowText(stredfilename);
 if(stredfilename.IsEmpty())
 return;
 CString temp,strtemp;strtemp=temp=strfilename;
 int pos=temp.Find("\\");
 while(pos>0)
 {
 temp=temp.Right(temp.GetLength()-1-pos);
 pos=temp.Find("\\");
 }
 strtemp=strtemp.Left(strtemp.GetLength()-temp.GetLength());
 strtemp+=stredfilename;
 CFile::Rename(strfilename,strtemp);//文件重命名
 AfxMessageBox("修改成功");
}

```

### 举一反三

根据本实例, 读者可以:

移动目录中的文件;

批量重命名文件。

## 5.6 文件的读取与保存

文件操作过程中经常会涉及文件的读取和保存。本节将通过顺序读取文件和制作日志文件等实例介绍文件读取和保存方面的知识。

## 实例 192 顺序读取文件

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\05\192

## 实例说明

本实例实现了按从前到后的顺序读取有规律的文本文件的功能, 有规律的文本文件如图 5.22 所示。运行本实例单击“打开文件”按钮打开有规律的文本文件, 单击“读取”按钮读取文本文件的一行, 并将读取的文本显示在编辑框中, 如图 5.23 所示。

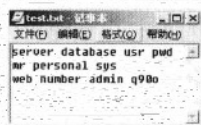


图 5.22 有规律的文本

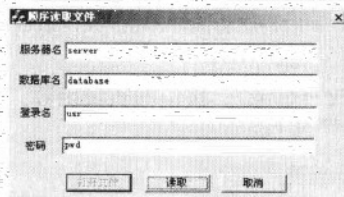


图 5.23 顺序读取文件

## 技术要点

本实例主要通过 CStdioFile 类的 ReadString 方法实现, 该方法用于一次读取文本文件的一行, 语法如下:

```
virtual LPTSTR ReadString(LPTSTR lpsz, UINT nMax);
BOOL ReadString(CString& rString);
```

参数说明:

- lpsz: 指向被读取文件的文件名字符串指针。
- nMax: 读取字符的数量。
- rString: 指向 CString 对象。

本实例的完成还需要结合 CString 对象的 Find 方法, 通过该方法可以找到分隔符的位置, 将分隔符左边的字符保存并显示, 将分隔符右边的字符保存, 并在其中继续查找分隔符的位置。

## 实现过程

(1) 新建名为 OrderReadFile 的对话框 MFC 工程。

(2) 在对话框上添加 4 个静态文本控件, 添加 4 个文本编辑框控件, 设置 ID 属性分别为 IDC\_EDSERVER、IDC\_EDDATABASE、IDC\_EDUSR 和 IDC\_EDPWD; 添加 3 个按钮控件, 设置 ID 属性分别为 IDC\_BTOPEN、IDC\_BTREAD 和 IDC\_BTEXIT, Caption 属性分别为“打开文件”、“读取”和“取消”。

(3) 在头文件 OrderReadFileDlg.h 加入变量声明:

```
CStdioFile file;
DWORD readlen;
```

(4) “读取”按钮的实现函数。该函数用于读取文本文件中的一行字符串, 代码如下:

```
void COrderReadFileDlg::OnRead()
{
 CString strserver, strdatabase, strusr, strpwd, readstring;
 if(!file)
 {
 if(readlen==file.GetLength())
 return;
 readstring=file.ReadString(readstring); //读取字符串
 int pos=readstring.Find("");
 strserver=readstring.Left(pos);
 }
}
```

```

readstring=readstring.Right(readstring.GetLength()-pos-1);
pos=readstring.Find(" "); //查找子串的位置
strdatabase=readstring.Left(pos);
readstring=readstring.Right(readstring.GetLength()-pos-1);
pos=readstring.Find(" ");
strusr=readstring.Left(pos);
readstring=readstring.Right(readstring.GetLength()-pos-1);

pos=readstring.Find(" ");
strpwd=readstring.Left(pos);
readstring=readstring.Right(readstring.GetLength()-pos-1);

GetDlgItem(IDC_EDSERVER)->SetWindowText(strserver);
GetDlgItem(IDC_EDDATABASE)->SetWindowText(strdatabase);
GetDlgItem(IDC_EDUSR)->SetWindowText(strusr);
GetDlgItem(IDC_EDPWD)->SetWindowText(strpwd);
}
}

```

(5) “打开文件”按钮的实现函数。该函数用于打开一个将要读取的文件，代码如下：

```

void COrderReadFileDlg::OnOpen()
{
 try{
 file.Open("test.txt",CFile::modeRead);
 GetDlgItem(IDC_BTOPEN)->EnableWindow(FALSE);
 }catch(CFileException *e)
 {
 TCHAR szBuf[256];
 e->GetErrorMessage(szBuf,256,NULL); //获取错误信息
 MessageBox(szBuf,T("Warning"));
 e->Delete();
 }
}

```

(6) “取消”按钮的实现函数。该函数用于实现窗体的关闭，代码如下：

```

void COrderReadFileDlg::OnExit()
{
 file.Close();
 this->OnCancel();
}

```

### 举一反三

根据本实例，读者可以：

- 将读取的文本信息保存到数据库；
- 将文本框中的数据写回文本文件。

## 实例 193 制作日志文件

本实例可以提高计算机安全性

实例位置：光盘\mingrisoft\05\193

### 实例说明

大多数应用软件都包含系统日志的功能，该日志可以记录操作及操作的时间。本实例实现读取日志内容并显示的功能，另外，还可以向日志文件中写入当前的系统时间，如图 5.24 所示。

### 技术要点

日志的读取主要通过 CStdioFile 类 ReadString 方法完成，通过一个循环语句将文本文件中的内容全部读取出来；日志的写入主要通过 FILE 对象的 fopen 方法和 fprintf 方法完成。fopen 方法用于打开目标文件，fprintf 方法用于向目标文件写入内容。

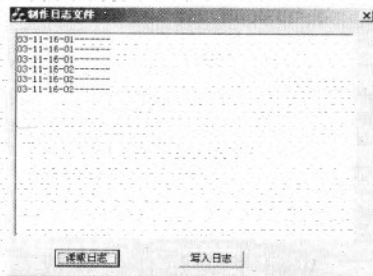


图 5.24 制作日志文件



## 实现过程

(1) 新建名为 LogFile 的对话框 MFC 工程。

(2) 在对话框上添加文本编辑框控件，设置 ID 属性为 IDC\_EDLOG，添加成员变量 m\_edlog；添加两个按钮控件，设置 ID 属性分别为 IDC\_BTREAD 和 IDC\_BTWRITE，设置 Caption 属性分别为“读取日志”和“写入日志”。

(3) “写入日志”按钮的实现函数。该函数用于实现向文件中写入日志信息，代码如下：

```
void CLogFileDlg::OnWrite()
{
 CTime time;
 time=CTime::GetCurrentTime();
 FILE* fp;
 fp=fopen("test.log","a");//打开文件
 fprintf(fp, "%s-----\n",time.Format("%d-%H-%M-%S"));//写入数据
 fclose(fp);//关闭文件
}
```

(4) “读取日志”按钮的实现函数。该函数用于将日志文件中的内容读取到编辑框中，代码如下：

```
void CLogFileDlg::OnRead()
{
 CString tmp,str;
 CStdioFile file;
 try{
 int i=file.Open("test.log",CFile::modeRead);//打开文件
 if(i==0)
 return;
 }catch(CFileException *e)
 {
 TCHAR szBuf[256];
 e->GetErrorMessage(szBuf,256,NULL);//读取错误信息
 MessageBox(szBuf,T("Warning"));
 e->Delete();//删除错误信息
 }
 while(1)
 {
 DWORD i=file.ReadString(str);//读取字符串
 if(i==0)goto end;//跳转标签
 tmp+=str;
 tmp+="\r\n";
 }
end:
 m_edlog.SetWindowText(tmp);
}
```

## 举一反三

根据本实例，读者可以：

- 制作数据库操作日志；
- 系统异常错误信息日志。

## 实例 194 获取 Word 文档属性

本实例可以提高计算机安全性

实例位置：光盘\mingrisoft\05\194

## 实例说明

Word 作为字处理软件，在 Windows 系统中被广泛使用。许多档案管理软件都是以 Word 格式组织的，因此，在程序中需要查看和维护 Word 文档。本例实现的功能是获取 Word 文档的属性，如图 5.25 所示。

## 技术要点

Word 文档属性的获取是通过复合文档操作接口 IPropertySetStorage 和 IPropertyStorage 来实现的。同时对应了

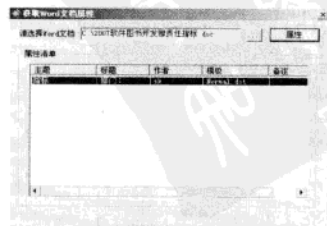


图 5.25 获取 Word 文档属性

属性读取所需要的结构信息的定义。

```
//定义属性结构
struct PropStrct{
 char* proName;
 long proFlag;
}Props[] = {
 {"Title",PIDS_I_TITLE}, //标题
 {"Subject",PIDS_I_SUBJECT}, //主题
 {"Author",PIDS_I_AUTHOR}, //作者
 {"Keywords",PIDS_I_KEYWORDS}, //关键字
 {"Comments",PIDS_I_COMMENTS}, //备注
 {"Template",PIDS_I_TEMPLATE}, //模板
 {"Last Author",PIDS_I_LASTAUTHOR}, //最后保存者
 {"Revision Number",PIDS_I_REVNUMBER}, //修订号
 {"Edit Time",PIDS_I_EDITTIME}, //编辑时间
 {"Last Printed",PIDS_I_LASTPRINTED}, //最后一次打印时间
 {"Created",PIDS_I_CREATE_DTM}, //创建日期
 {"Last Saved",PIDS_I_LASTSAVE_DTM}, //最后一次保存日期
 {"Page Count",PIDS_I_PAGECOUNT}, //页数
 {"Word Count",PIDS_I_WORDCOUNT}, //字数
 {"Char Count",PIDS_I_CHARCOUNT}, //字符数
 {"Thumb nail",PIDS_I_THUMBNAI},
 {"AppName",PIDS_I_APPNAME}, //应用程序名称
 {"Doc Security",PIDS_I_DOC_SECURITY}, //安全性
 {0,0}
};
```

## 实现过程

(1) 新建名为 GetProp 的对话框 MFC 工程。

(2) 在对话框上添加一个静态文本控件，设置其标题名称为“请选择 Word 文档”；添加一个文本编辑控件；添加两个按钮控件和一个列表视图控件。

(3) 主要程序代码。

```
void CGetPropDlg::OnProperty()
{
 UpdateData(TRUE);
 if (m_FileName.IsEmpty())
 {
 MessageBox("请选择Word文档");
 return;
 }
 m_PropertyList.DeleteAllItems();
 IStorage* pStorage = NULL;
 IPropertySetStorage* pSetprop = NULL;

 WCHAR filename[MAX_PATH];

 wcsncpy(filename,m_FileName.AllocSysString());
 //打开复合文件
 ::StgOpenStorage(filename,NULL,STGM_READ|STGM_SHARE_EXCLUSIVE,NULL,0,
 &pStorage);

 if (pStorage==NULL)
 {
 MessageBox("打开文件错误");
 return;
 }
 //获取属性接口
 pStorage->QueryInterface(IID_IPropertySetStorage,(void**)&pSetprop);

 if (pSetprop==NULL)
 {
 MessageBox("获取属性错误");
 return;
 }
 IPropertyStorage* pPropStorage = NULL;

 pSetprop->Open(FMTID_SummaryInformation,STGM_READ|STGM_SHARE_EXCLUSIVE,&pPropStorage);

 if (pPropStorage==NULL)
 {
 MessageBox("获取属性错误");
 }
}
```

```

 return;
 }

 //定义属性结构
 struct PropStrct{
 char* proName;
 long proFlag;
 }Props[] = {
 {"Title",PIDSI_TITLE}, //标题
 {"Subject",PIDSI_SUBJECT}, //主题
 {"Author",PIDSI_AUTHOR}, //作者
 {"Keywords",PIDSI_KEYWORDS}, //关键字
 {"Comments",PIDSI_COMMENTS}, //备注
 {"Template",PIDSI_TEMPLATE}, //模板
 {"Last Author",PIDSI_LASTAUTHOR}, //最后保存者
 {"Revision Number",PIDSI_REVNUMBER}, //修订号
 {"Edit Time",PIDSI_EDITTIME}, //编辑时间
 {"Last Printed",PIDSI_LASTPRINTED}, //最后一次打印时间
 {"Created",PIDSI_CREATE_DTM}, //创建日期
 {"Last Saved",PIDSI_LASTSAVE_DTM}, //最后一次保存日期
 {"Page Count",PIDSI_PAGECOUNT}, //页数
 {"Word Count",PIDSI_WORDCOUNT}, //字数
 {"Char Count",PIDSI_CHARCOUNT}, //字符数
 {"Thumb nai",PIDSI_THUMBNAI}, //应用程序名称
 {"AppName",PIDSI_APPNAME}, //安全性
 {"Doc Security",PIDSI_DOC_SECURITY},
 {0,0}
 };

 int PropCount = sizeof(Props)/sizeof(PropStrct)-1;

 PROPSPEC * pPropSpec = new PROPSPEC[PropCount];
 PROPVARIANT* pPropVariant = new PROPVARIANT[PropCount];

 for (int i =0; i<PropCount; i++)
 {
 ZeroMemory(&pPropSpec[i],sizeof(PropStrct));
 pPropSpec[i].ulKind = PRSPEC_PROPID;
 pPropSpec[i].propid = Props[i].proFlag;
 }
 //读取属性信息
 HRESULT hr = pPropStorage->ReadMultiple(PropCount,pPropSpec,pPropVariant);

 if (FAILED(hr))
 {
 MessageBox("获取属性失败");
 }
 else
 {
 m_PropertyList.InsertItem(0,"");
 for (int m = 0; m<PropCount; m++)
 {
 if (pPropVariant[m].vt==VT_LPSTR)
 {
 char* pbstr = (char*)pPropVariant[m].bstrVal;

 switch (m)
 {
 case 1:
 m_PropertyList.SetItemText(0,0,pbstr);
 break;
 case 0:
 m_PropertyList.SetItemText(0,1,pbstr);
 break;
 case 2:
 m_PropertyList.SetItemText(0,2,pbstr);
 break;
 case 5:
 m_PropertyList.SetItemText(0,3,pbstr);
 break;
 case 4:
 m_PropertyList.SetItemText(0,4,pbstr);
 break;
 }
 }
 }
 }

```

```

 }
 delete [] pPropSpec;
 delete [] pPropVariant;

 pPropStorage->Release(); //接口释放
 pSetprop->Release();
 pStorage->Release();
}

void CGetPropDlg::OnBrown()
{
 CFileDialog fDlg(TRUE, "doc", NULL, OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, "Word文档|*.doc", this);
 if (fDlg.DoModal() == IDOK)
 {
 m_FileName = fDlg.GetPathName();
 UpdateData(FALSE);
 }
}

```

### 举一反三

根据本实例，读者可以：

- 将读取的 Word 文本信息保存到数据库；
- 将文本框中的数据写入到 Word 文档中。

## 实例 195 将 Word 转换为 HTML

这是一个自娱自乐的实例

实例位置：光盘\mingrisoft\05\195

### 实例说明

CHM 格式的帮助文档是通过 HTML 格式的文档生成的，但在编辑这些资料时可能是通过 Word 文档进行编辑的，所以本实例将向读者介绍如何将 Word 格式的文档转换成 HTML 格式的文档，如图 5.26 所示。

### 技术要点

将 Word 文档转成 HTML 文件需要将 office 文件夹下的 MSWORD9.OLB 文件中的类接口通过 Add Class From a type library 方式加入到工程中。再通过 \_Application 接口创建 Word 工程实例，根据此接口获取 \_Document 文档接口，根据文档接口的 SaveAs 方法将 Word 文档另存为 HTML 格式的文档。

### 实现过程

- (1) 新建一个基于对话框 MFC 工程。
- (2) 在对话框上添加一个静态文本控件，设置标题为“打开 Word 文档”；添加一个文本编辑控件，用来设置 Word 文档的路径；添加两个按钮控件，分别设置标题为“转换”和“退出”。
- (3) 主要程序代码。

```

void CWordToTxtDlg::OnConvert()
{
 COleVariant covOptional((long)DISP_E_PARAMNOTFOUND, VT_ERROR);
 CString strfile;
 CString strtxt, strext;
 GetDlgItem(IDC_EDFILE)->GetWindowText(strfile);
 _Application app;
 Documents docs;
 _Document doc;
 COleVariant tmplt(strfile);
 COleVariant newtmp(short(false));
 COleVariant type(short(0));

```



图 5.26 将 Word 转换为 HTML



```

COleVariant vble(short(true));
COleVariant format(short(8));
app.CreateDispatch("word.Application");
docs.AttachDispatch(app.GetDocuments());
doc=docs.Add(&tmplt,&newtmp,&type,&vble);

strest=GetFileExt(strfile);//获取文件扩展名
strtxt=strfile.Left(strfile.GetLength()-strest.GetLength());
if(strtxt.Right(1)!=".")
strtxt+="html";
COleVariant file(strtxt);
//文件另存
doc.SaveAs(file,format,covOptional,covOptional,covOptional,covOptional,
covOptional,covOptional,covOptional,covOptional,covOptional);
app.Quit(&COleVariant(short(false)),&COleVariant(short(0)),&COleVariant(short(false)));
docs.ReleaseDispatch();
doc.ReleaseDispatch();
app.ReleaseDispatch();
}

```

## 举一反三

根据本实例，读者可以：

- 将 Word 文档转成文本文件；
- 将 Word 文档转成其他格式的文件。

## 实例 196 提取 Word 文档目录

本实例可以提高计算机安全性

实例位置：光盘\mingrisoft\05\196

## 实例说明

Word 的核心组件是 ActiveX 组件，任何一种编程工具都可以通过这个组件控制 Word 文档。Word 目录提取工具也是通过调用 ActiveX 组件中的类实现 Word 文档中目录的提取的。如图 5.27 所示。

## 技术要点

本实例通过 Word 组件来完成，需要将 office 文件夹下的 MSWORD9.OLB 文件中的类接口通过 Add Class From a type library 方式加入到工程中。此时工程中将生成引用 Word 组件所需要的头文件，通过该头文件中定义的类型即可对 Word 文档进行操作。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个列表框控件，用来显示从 Word 文档中提取出来的目录。添加 3 个按钮控件，分别设置其 Caption 属性为“提取”、“保存到文件”和“取消”。
- (3) 在窗体中单击“保存到文件”按钮，将打开一个 Word 文档并对文档中的目录进行提取，并显示在列表框控件中。实现代码如下：

```

void CFetchDirDlg::OnFetch()
{
 //选择word文档
 CFileDialog fDlg(TRUE, "", "", OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
 "word文档[*.doc]");
 if (fDlg.DoModal() != IDOK)
 {
 CString szFileName = fDlg.GetPathName();//获取Word文档路径
 _Application wordApp;

```



图 5.27 提取 Word 文档目录

```

Documents wordDocs;
wordApp.CreateDispatch("word.Application");//创建Word工程
wordDocs.AttachDispatch(wordApp.GetDocuments());//获取Word文档
Document wordDoc;

CComVariant filename(szFileName.AllocSysString()), visible(TRUE), doctype(0), doctemplate(TRUE);
szFileName.ReleaseBuffer();
//关联Word打开的文档
wordDoc.AttachDispatch(wordDocs.Add(&filename, &visible, &doctype, &doctemplate));

Paragraphs pgraphs;
pgraphs.AttachDispatch(wordDoc.GetParagraphs());

m_List.ResetContent(); //删除列表中的内容

long pgraphCount = pgraphs.GetCount();
for (long i = 1; i <= pgraphCount; i++)
{
 Paragraph pgraph;
 pgraph.AttachDispatch(pgraphs.Item(i));
 Range pragRange;
 pragRange.AttachDispatch(pgraph.GetRange());
 ParagraphFormat format;
 format.AttachDispatch(pragRange.GetParagraphFormat());
 CComVariant value;

 Style style;

 value = format.GetStyle();//获取样式
 style.AttachDispatch(value.pdispVal);

 CString szHeaderName = style.GetNameLocal();
 char szName[10] = {0};
 strncpy(szName, szHeaderName.GetBuffer(0), 6);
 szHeaderName.ReleaseBuffer(0);
 if (strcmp(szName, "标题 1") == 0)
 {
 //读取标题内容
 CString szText = pragRange.GetText();
 int nIndex = m_List.AddString(szText);
 m_List.SetItemData(nIndex, 1);
 }
 else if (strcmp(szName, "标题 2") == 0)
 {
 //读取标题内容
 CString szText = pragRange.GetText();
 int nIndex = m_List.AddString(szText);
 m_List.SetItemData(nIndex, 2);
 }
 else if (strcmp(szName, "标题 3") == 0)
 {
 //读取标题内容
 CString szText = pragRange.GetText();
 int nIndex = m_List.AddString(szText);
 m_List.SetItemData(nIndex, 3);
 }
}

CComVariant save(0), format(0), route(0);
wordDoc.Close(&save, &format, &route);//关闭文档
wordApp.Quit(&save, &format, &route); //关闭工程
}
}

```

(4) 在窗体中单击“保存到文件”按钮，将列表框控件中显示的目录保存到指定的文件中。  
实现代码如下：

```

//保存列表中的内容到TXT文件中
void CFetchDirDlg::OnSaveToFile()
{
 CFileDialog fIDlg(FALSE, "txt", "directory.txt", OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
 "文本文件*.txt|");
 if (fIDlg.DoModal() == IDOK)
 {
 CString szSaveName = fIDlg.GetPathName();
 CFile file;
 file.Open(szSaveName, CFile::modeCreate | CFile::modeReadWrite);
 int nLineCount = m_List.GetCount();
 }
}

```

```
DWORD dwItemData = 0;
char szReturn[2] = {"\n"}; //换行符
char szIndent[5] = {' ', ' ', ' ', ' '}; //缩进4个空格
for (int i=0; i<nLineCount; i++)
{
 CString szText;
 m_List.GetText(i, szText);
 dwItemData = m_List.GetItemData(i);
 if (dwItemData == 1) //1级目录
 {
 file.Write(szText.GetBuffer(0), szText.GetLength());
 file.Write(szReturn, 2);
 }
 else if (dwItemData == 2) //2级目录
 {
 for(int j = 0; j< dwItemData-1; j++)
 {
 file.Write(szIndent, 5);
 }
 file.Write(szText.GetBuffer(0), szText.GetLength());
 file.Write(szReturn, 2);
 }
 else if (dwItemData == 3) //3级目录
 {
 for(int j = 0; j< dwItemData-1; j++)
 {
 file.Write(szIndent, 5);
 }
 file.Write(szText.GetBuffer(0), szText.GetLength());
 file.Write(szReturn, 2);
 }

 szText.ReleaseBuffer();
}
file.Close();
}
```

### 举一反三

根据本实例，读者可以：

- 提取 word 中的任意目录；
- 通过文本文件生成 Word 目录。

## 5.7 文件管理

文件管理包括整理磁盘文件、修改及获取文件的属性、磁盘空间预警等。本节中通过几个实例程序分别介绍这几方面的知识，使读者能够真正开发出文件管理方面的实用软件，迅速提高编程技能。

### 实例 197 分类整理磁盘文件

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\05\197

#### 实例说明

本实例实现将一个目录中的相同扩展名的文件移动到其  
他目录中。通过“...”按钮选择要整理的目录，打开目录以  
后，程序会将目录中的文件全部显示在列表中，通过选择文  
件过滤中不同的文件扩展名，实现对相同扩展名文件的提取，  
相同扩展名的文件将显示在列表框中，最后通过“移动结果  
到”按钮来将列表中的文件复制到指定的目录。程序运行结  
果如图 5.28 所示。

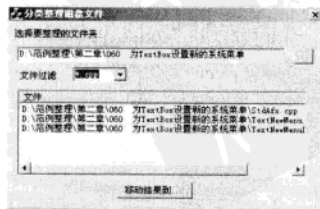


图 5.28 分类整理磁盘文件

## 技术要点

本实例通过自定义函数 TidyFile 实现分类整理磁盘文件的功能，该函数调用 CFileFind 类的 FindFile 方法和 FindNextFile 方法实现文件的查找，通过查找可以将相同扩展名的文件显示在列表中。

## 实现过程

(1) 新建名为 TidyFile 的对话框 MFC 工程。

(2) 在对话框上添加两个静态文本控件；添加文本编辑控件，设置 ID 属性为 IDC\_EDDIR，添加组合框控件，设置 ID 属性为 IDC\_COFILEEXT，添加成员变量 m\_fileext；添加列表视图控件，设置 ID 属性为 IDC\_FILELIST，添加成员变量 m\_filelist；添加按钮控件，设置 ID 属性为 IDC\_BTSAVE，设置 ID 属性为“移动结果到...”。

(3) 在头文件 TidyFileDialog.h 加入如下变量声明：

CString strpath;

(4) 在 OnInitDialog 中初始化列表控件，代码如下：

```
BOOL CTidyFileDialog::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_filelist.SetExtendedStyle(LVS_EX_GRIDLINES);
 m_filelist.InsertColumn(0, "文件", LVCFMT_LEFT, 450);
 m_fileext.AddString("*.cpp");
 m_fileext.AddString("*.h");
 return TRUE;
}
```

(5) “...”按钮的实现函数。该函数用于设置需要整理文件所在文件夹，代码如下：

```
void CTidyFileDialog::OnAdd()
{
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer, MAX_PATH);
 bi.hwndOwner = GetSafeHwnd();
 bi.pidlRoot = NULL;
 bi.pszDisplayName = buffer;
 bi.lpszTitle = "选择一个文件夹";
 bi.ulFlags = BIF_EDITBOX;
 bi.lpfnc = NULL;
 bi.lParam = 0;
 bi.ilImage = 0;
 LPITEMIDLIST pList = NULL;
 if((pList = SHBrowseForFolder(&bi)) != NULL)
 {
 char path[MAX_PATH];
 ZeroMemory(path, MAX_PATH);
 SHGetPathFromIDList(pList, path);
 GetDlgItem(IDC_EDDIR) -> SetWindowText(path);
 strpath = path;
 TidyFile(path, ".*");
 }
}
```

(6) 自定义函数 TidyFile，用来查找特定文件夹下的文件，代码如下：

```
void CTidyFileDialog::TidyFile(CString path, CString strext)
{
 if(path.Right(1) != "\\")
 path += "\\";
 path += strext;
 CFileFind fd;
 BOOL bfind = fd.FindFile(path); // 查找文件
 int i = 0;
 while(bfind)
 {
 bfind = fd.FindNextFile(); // 查找下一个文件
 if(!fd.IsDirectory() && !fd.IsDots())
 m_filelist.InsertItem(i, fd.GetFilePath(), 0);
 i++;
 }
}
```



(7) “移动结果到...”按钮的实现函数。该函数用于将列表控件中的文件移动到指定文件夹下,代码如下:

```
void CTidyFileDlg::OnSave()
{
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer, MAX_PATH);
 bi.hwndOwner = GetSafeHwnd();
 bi.pidlRoot = NULL;
 bi.pszDisplayName = buffer;
 bi.lpszTitle = "选择一个文件夹";
 bi.ulFlags = BIF_EDITBOX;
 bi.lpfn = NULL;
 bi.lParam = 0;
 bi.ilImage = 0;
 LPITEMIDLIST pList = NULL;
 if (pList = SHBrowseForFolder(&bi)) != NULL
 {
 char path[MAX_PATH];
 ZeroMemory(path, MAX_PATH);
 SHGetPathFromIDList(pList, path);
 int count = m_filelist.GetItemCount();
 CString strfullname, strpathoname, stronlyname, strdesname;
 for (int i = 0; i < count; i++)
 {
 stronlyname = strfullname = m_filelist.GetItemText(i, 0);
 int pos = stronlyname.Find("\\");
 while (pos > 0)
 {
 stronlyname = stronlyname.Right(stronlyname.GetLength() - 1 - pos);
 pos = stronlyname.Find("\\");
 }
 strdesname.Format("%s\\%s", path, stronlyname);
 ::MoveFile(strfullname, strdesname);
 }
 }
}
```

(8) 控件 IDC\_COFILEEXT 的 CBN\_SELCHANGE 消息的实现函数。该函数通过获得用户对文件类型的选择对文件进行过滤,代码如下:

```
void CTidyFileDlg::OnSelchangeCofileext()
{
 m_filelist.DeleteAllItems();
 CString strext;
 int cursel = m_fileext.GetCurSel();
 m_fileext.GetLBText(cursel, strext);
 TidyFile(strpath, strext);
}
```

### 举一反三

根据本实例,读者可以:

- 分类整理磁盘中的图片文件;
- 分类存储磁盘中的音频文件。

## 实例 198 计算机磁盘空间报警程序

这是一个自娱自乐的实例

实例位置: 光盘\mingrisoft\05\198

### 实例说明

如果系统磁盘的空间不足,将会影响计算机的运行速度。本实例是一个磁盘空间报警的程序,首先通过列表显示系统所有磁盘分区及每个磁盘分区的总空间和可用空间,然后对单个磁盘空间进行监控并报警,在组合框中选择进行监控的磁盘分区,在文本框中输入监控的数值,如图 5.29 所示。单击“设置”按钮程序将在系统托盘中运行,当磁盘空间小于设置值时,程序

将弹出对话框提示用户。

## 技术要点

本实例主要通过 GetLogicalDriveStrings 函数和 GetDiskFreeSpaceEx 函数的使用实现磁盘空间报警功能的，GetLogicalDriveStrings 函数主要实现系统磁盘驱动器的查询，其语法如下：

DWORD GetLogicalDriveStrings(DWORD nBufferLength, LPTSTR lpBuffer);

参数说明：

- nBufferLength: lpBuffer 参数所指向的缓冲区的大小。
- lpBuffer: 接收数据的缓冲区，接收的数据中包含磁盘所有系统磁盘驱动器信息，例如系统中包含 C 盘，D 盘，E 盘，接收的数据就是 c:\<null>d:\<null>e:\<null><null>。

GetDiskFreeSpaceEx 函数用来查询系统磁盘驱动器总空间、可用空间和已用空间，其语法如下：

BOOL GetDiskFreeSpaceEx(LPCTSTR lpDirectoryName,  
PULARGE\_INTEGER lpFreeBytesAvailableToCaller,  
PULARGE\_INTEGER lpTotalNumberOfBytes,  
PULARGE\_INTEGER lpTotalNumberOfFreeBytes);

参数说明：

- lpDirectoryName: 磁盘驱动器名称。
- lpFreeBytesAvailableToCaller: 磁盘使用空间。
- lpTotalNumberOfBytes: 磁盘总空间。
- lpTotalNumberOfFreeBytes: 磁盘可用空间。

注意：GetDiskFreeSpaceEx 返回的磁盘空间大小是以字节为单位的。

## 实现过程

(1) 新建名为 DiakWarn 的对话框 MFC 工程。

(2) 在对话框上添加两个静态文本控件；添加列表视图控件，设置 ID 属性为 IDC\_DISKLIST，添加成员变量 m\_disklist；添加组合框控件，设置 ID 属性为 IDC\_CMBSPACE，添加成员变量 m\_space；添加两个按钮控件，设置 ID 属性分别为 IDC\_BTSET 和 IDC\_BTEXIT，设置 Caption 属性分别为“设置”和“退出”。

(3) 主要程序代码。

```

BOOL CDiskWarnDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 //此处代码省略
 m_disklist.SetExtendedStyle(LVS_EX_GRIDLINES);
 m_disklist.InsertColumn(0,"磁盘驱动器",LVCFMT_LEFT,150);
 m_disklist.InsertColumn(1,"驱动器大小",LVCFMT_LEFT,150);
 m_disklist.InsertColumn(2,"可用空间",LVCFMT_LEFT,150);
 imglist.Create(16,16,ILC_COLOR32,ILC_MASK,0,0);
 imglist.Add((AfxGetApp()->LoadIcon(IDI_DISK)));
 m_disklist.SetImageList(&imglist,LVSIL_SMALL);
 DWORD size;
 size=::GetLogicalDriveStrings(0,NULL);
 if(size!=0)
 {
 HANDLE heap=::GetProcessHeap();
 LPSTR lp=(LPSTR)HeapAlloc(heap,HEAP_ZERO_MEMORY,size*sizeof(TCHAR));
 ::GetLogicalDriveStrings(size*sizeof(TCHAR),lp);//获取驱动器信息
 int i=0;
 while(*lp!=0)
 {
 m_disklist.InsertItem(i,lp,0);
 m_space.AddString(lp);
 lp=_tcschr(lp,0)+1;
 i++;
 }
 }
}

```

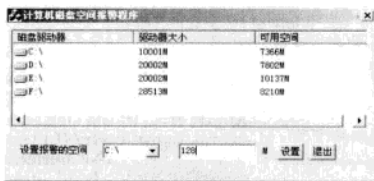


图 5.29 计算机磁盘空间报警程序

```

int num=m_disklist.GetItemCount();
for(int i=0;i<num;i++)
{
 CString str,temp;
 str=m_disklist.GetItemText(i,0);
 ::GetDiskFreeSpaceEx(str,&freespace,&totalsize,&availablesize);//获取磁盘空间
 temp.Format("%ldM",totalsize.QuadPart/1048576);
 m_disklist.SetItemText(i,1,temp);
 temp.Format("%ldM",freesize.QuadPart/1048576);
 m_disklist.SetItemText(i,2,temp);
}
return TRUE;
}
}

```

(4) “设置”按钮的实现函数。该函数用于设置预警的磁盘空间，代码如下：

```

void CDiskWarnDlg::OnSet()
{
 GetDlgItem(IDC_EDSPACE)->GetWindowText(strwarn);
 strfreespace=m_disklist.GetItemText(sel,2);
 if(atoi(strfreespace)<atoi(strwarn))
 {
 AfxMessageBox("设置无效");
 return;
 }
 SetTimer(1,1000,NULL);
 GetDlgItem(IDC_BTSET)->EnableWindow(FALSE);
 GetDlgItem(IDC_EDSPACE)->EnableWindow(FALSE);
 GetDlgItem(IDC_CMBSPACE)->EnableWindow(FALSE);
 ShowWindow(SW_HIDE);
}

```

(5) 控件 IDC\_CMBSPACE 的 CBN\_SELCHANGE 消息的实现函数，代码如下：

```

void CDiskWarnDlg::OnSelchangeCmbpace()
{
 sel=m_space.GetCurSel();
}

```

(6) 对话框中 WM\_TIMER 消息的实现函数，代码如下：

```

void CDiskWarnDlg::OnTimer(UINT nIDEvent)
{
 ULARGE_INTEGER availablesize;
 int num=m_disklist.GetItemCount();
 for(int i=0;i<num;i++)
 {
 CString str,temp;
 str=m_disklist.GetItemText(i,0);
 ::GetDiskFreeSpaceEx(str,&freespace,&totalsize,&availablesize);//获取磁盘空间
 temp.Format("%ldM",freesize.QuadPart/1048576);
 m_disklist.SetItemText(i,2,temp);
 }
 Invalidate();
 GetDlgItem(IDC_EDSPACE)->GetWindowText(strwarn);
 strfreespace=m_disklist.GetItemText(sel,2);
 if(atoi(strfreespace)<atoi(strwarn))
 {
 AfxMessageBox("磁盘空间已超出设置");
 KillTimer(1);
 }
 CDialog::OnTimer(nIDEvent);
}

```

## 举一反三

根据本实例，读者可以：

- 在安装应用软件的时候，事先检测磁盘的剩余空间；
- 在保存文件的时候，判断磁盘剩余空间的容量是否大于文件的大小。

## 实例 199 批量改变指定文件的属性

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\05\199

## 实例说明

在通常情况下，在文件的属性窗口中可以修改文件的属性。本实例可以批量修改指定文

### 技术要点

文件

|                 |                                   |
|-----------------|-----------------------------------|
| D:\内网整理\第五季\107 | 批量改变指定文件的属性\StdFile.h             |
| D:\内网整理\第五季\107 | 批量改变指定文件的属性\ChangeFileAttrDlg.h   |
| D:\内网整理\第五季\107 | 批量改变指定文件的属性\StdFile.txt           |
| D:\内网整理\第五季\107 | 批量改变指定文件的属性\Resource.h            |
| D:\内网整理\第五季\107 | 批量改变指定文件的属性\StdFile.cpp           |
| D:\内网整理\第五季\107 | 批量改变指定文件的属性\ChangeFileAttrDlg.cpp |

添加文件    ☐ 只读    ☐ 系统    ☒ 隐藏    设置

图 5.30 批量改变指定文件的属性

-  注意：调用 SetStatus 方法修改文件属性前一定要调用 GetStatus 方法先来获得文件的属性，然后在获得的 CFileStatus 数据结构对象中追加属性值。

## 实现过程

- (3) 主要程序代码:

```

BOOL CChangeMFileAttrDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 //此处代码省略
 m_filelist.SetExtendedStyle(LVS_EX_GRIDLINES);
 m_filelist.InsertColumn(0,"文件",LVCFMT_LEFT,450);
 return TRUE;
}

```

- (4) “添加文件”按钮的实现函数，代码如下：

```
void CChangeMFileAttrDlg::OnAddFile()
{
 CFileDialog log(TRUE, "文件", "*", OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT|
 OFN_ALLOWMULTISELECT, "FILE(*.*)|*.jpg|*.jpeg(*.jpg)*.jpeg|
 文本(*.txt)|*.txt|", NULL);
 if(log.DoModal()==IDOK)
 {
 POSITION pos = log.GetStartPosition();
 while(pos != NULL)
 {
 CString pathname=log.GetNextPathName(pos);
 m_filelist.InsertItem(m_filelist.GetItemCount(),pathname);
 }
 }
}
```



(5) “设置”按钮的实现函数,代码如下:

```
void CChangeMFileAttrDlg::OnSetFile()
{
 BYTE m_newattri;
 CFileStatus status;
 if(m_chreadonly.GetCheck())
 m_newattri|=0x01;
 if(m_chsystem.GetCheck())
 m_newattri|=0x04;
 if(m_chhide.GetCheck())
 m_newattri|=0x02;
 int count=m_filelist.GetItemCount();
 for(int i=0;i<count;i++)
 {
 CFile::GetStatus(m_filelist.GetItemText(i,0),status);
 status.m_attribute=m_newattri;
 CFile::SetStatus(m_filelist.GetItemText(i,0),status);
 }
}
```

### 举一反三

根据本实例,读者可以:

- 在整理文件时设置文件的属性;
- 在文件管理程序中查看文件的属性。

## 5.8 加密与解密

本节包含两部分内容,即对文件的加密与解密和对文件夹的加密与解密,下面通过几个具体实例来讲解如何实现文件及文件夹的加密和解密。

### 实例 200 文件的加密与解密

这是一个可以提高分析能力的实例

实例位置: 光盘\mingrisoft\05\200

### 实例说明

文件的加密与解密技术非常重要,如可以将某些机密文件或不想被人所知的文件进行加密保存,待查看时再按照设置的密钥进行解密,查看其明文文件。本实例完成对文本文件的加密。运行程序,先打开一个文本文件,如图 5.31 所示,然后通过“加密”按钮进行加密,通过“解密”按钮进行解密。

### 技术要点

本实例通过 CFile 类的 Read 方法读取文件,然后对每个字节和固定数“2”进行异或运算,最后将运算的结果通过 Write 方法写入文件。

### 实现过程

- (1) 新建名为 FileEncry 的对话框 MFC 工程。
- (2) 在对话框上添加 1 个文本编辑控件,设置 ID 属性为 IDC\_EDBODY,添加 3 个按钮控件,设置 ID 属性分别为 IDC\_BTOPEN, IDC\_BTENCRY, IDC\_BTUNENCRY,设置 Caption 属性分别为“打开文件”,“加密”,“解密”。
- (3) 在头文件 FileEncryDlg.h 加入变量声明:

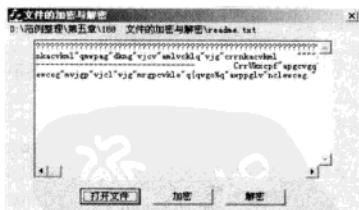


图 5.31 文件的加密与解密

CString strpathtemp;

(4) “打开文件”按钮的实现函数，该函数用于添加需要加密内容的文本文件，代码如下：

void CFileEntryDlg::OnOpen()

```
{
 CFileDialog log(TRUE, "文件", NULL, OFN_HIDEREADONLY, "FILE(*.txt)|*.txt|", NULL);
 if(log.DoModal() != IDOK)
 {
 CString tmp, str, path;
 path = log.GetPathName();
 strpathtemp = log.GetFileName();
 int pos = path.Find(strpathtemp);
 strpathtemp = path.Left(pos);
 CStdioFile file;
 try{
 int i = file.Open(path, CFile::modeRead); //以只读方式打开文件
 if(i == 0)
 {
 return;
 }
 } catch(CFileException *e)
 {
 TCHAR szBuf[256];
 e->GetErrorMessage(szBuf, 256, NULL);
 MessageBox(szBuf, _T("Warning"));
 e->Delete();
 }
 while(1)
 {
 DWORD i = file.ReadString(str); //从文件中读取数据
 if(i == 0) goto end;
 tmp += str;
 tmp += "\r\n";
 }
 end:
 GetDlgItem(IDC_EDBODY)->SetWindowText(tmp);
 m_filepath.SetWindowText(path);
}
}
```

(5) “加密”按钮的实现函数，该函数用于对文本内容进行加密，代码如下：

void CFileEntryDlg::OnEncrypt()

```
{
 CString path, desname;
 m_filepath.GetWindowText(path);
 if(path.IsEmpty()) return;
 desname.Format("%smingrisofttemp.txt", strpathtemp);
 //创建文件
 HANDLE handle = ::CreateFile(desname, GENERIC_WRITE, FILE_SHARE_WRITE, 0,
 CREATE_NEW, FILE_ATTRIBUTE_NORMAL, NULL);
 if(handle) ::CloseHandle(handle);
 CFile readfile, writefile;
 int i = readfile.Open(path, CFile::modeRead); //打开文件
 writefile.Open(desname, CFile::modeCreate | CFile::modeReadWrite); //打开文件
 if(i == 0) return;
 char buf[128];
 char desbuf[128];
 while(1)
 {
 ZeroMemory(buf, 128);
 ZeroMemory(desbuf, 128);
 DWORD i = readfile.Read(buf, 128); //读取数据
 for(int p = 0; p < i; p++)
 {
 char m = buf[p];
 desbuf[p] = m ^ 2;
 }
 writefile.Write(desbuf, i); //写入数据
 if(i == 0) goto end;
 }
 end:
 readfile.Close(); //关闭文件
 writefile.Close();
 ::DeleteFile(path);
 ::rename(desname, path);
 AfxMessageBox("加密完成");
}
}
```

(6) “解密”按钮的实现函数，该函数用于对文本内容进行解密，代码如下：

```

void CFileEncryDlg::OnUnEncry()
{
 CString path,desname;
 m_filepath.GetWindowText(path);
 if(path.IsEmpty())return;
 desname.Format("%smingrisofttemp.txt",strpathtemp);
 //创建文件
 HANDLE handle=::CreateFile(desname,GENERIC_WRITE,FILE_SHARE_WRITE,0,
 CREATE_NEW,FILE_ATTRIBUTE_NORMAL,NULL);
 if(handle)::CloseHandle(handle);
 CFile readfile,writefile;
 int i=readfile.Open(path,CFile::modeRead);//打开文件
 writefile.Open(desname,CFile::modeCreate|CFile::modeReadWrite);//打开文件
 if(i==0)return;
 char buf[128];
 char desbuf[128];
 while(1)
 {
 ZeroMemory(buf,128);
 ZeroMemory(desbuf,128);
 DWORD i=readfile.Read(buf,128);//读取数据
 for(int p=0;p<1;p++)
 {
 char m=buf[p];
 desbuf[p]=m^2; //加密过程
 }
 writefile.Write(desbuf,i);//写入数据
 if(i==0)goto end;
 }
 end:
 readfile.Close();//关闭文件
 writefile.Close();
 ::DeleteFile(path);
 ::rename(desname,path);
 AfxMessageBox("解密完成");
}

```

## 举一反三

根据本实例，读者可以：

- 完成中文文件的加密与解密；
- 通过图形加密与解密文件。

## 实例 201 文件夹加密

这是一个可以提高分析能力的实例

实例位置：光盘\mingrisoft\05\201

### 实例说明

通过加密文件夹的方法可以防止文件夹被打开，本实例便实现了这一功能。运行程序，选择将要加密的文件夹，如图 5.32 所示，单击“加密”按钮，就可以将该文件夹加密，对已经加密过的文件夹，单击“解密”按钮可以进行解密。

### 技术要点

利用 rename 函数将选中的文件夹名称修改为原文件夹名称+{d6277990-4c6a-11cf-8d87-00aa0060f5bf}，例如：“My eBook.{d6277990-4c6a-11cf-8d87-00aa0060f5bf}”，就完成对文件夹的加密。

若要解密文件夹，可以利用 rename 函数将文件夹名称中的 “. {d6277990-4c6a-11cf-8d87-00aa0060f5bf}” 字符去除就可以了。

### 实现过程

(1) 新建名为 FolderEncry 的对话框 MFC 工程。

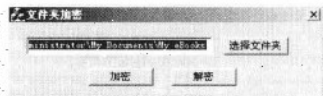


图 5.32 文件夹加密

(2) 在对话框上添加文本编辑控件, 设置 ID 属性为 IDC\_EDDIR, 添加 3 个按钮控件, 设置 ID 属性分别为 ID\_ADD、IDC\_BTENCRY 和 IDC\_BTUNENCRY, 设置 Caption 属性分别为“选择文件夹”、“加密”和“解密”。

(3) “选择文件夹”按钮的实现函数, 该函数用于选择需要加密的文件夹, 代码如下:

```
void CFolderEncryDlg::OnAdd()
{
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer, MAX_PATH);
 bi.hwndOwner = GetSafeHwnd();
 bi.pidlRoot = NULL;
 bi.pszDisplayName = buffer;
 bi.lpszTitle = "选择一个文件夹";
 bi.ulFlags = BIF_EDITBOX;
 bi.lpfnc = NULL;
 bi.lParam = 0;
 bi.ilImage = 0;
 LPITEMIDLIST pList = NULL;
 if((pList = SHBrowseForFolder(&bi)) != NULL)
 {
 char path[MAX_PATH];
 ZeroMemory(path, MAX_PATH);
 SHGetPathFromIDList(pList, path);
 GetDlgItem(IDC_EDDIR) -> SetWindowText(path);
 }
}
```

(4) “加密”按钮的实现函数, 该函数用于加密文件夹, 代码如下:

```
void CFolderEncryDlg::OnEncry()
{
 CString strpath, despath;
 GetDlgItem(IDC_EDDIR) -> GetWindowText(strpath);
 despath.Format("%s.{d6277990-4c6a-11cf-8d87-00aa0060f5bf}", strpath);
 ::rename(strpath, despath);
}
```

(5) “解密”按钮的实现函数, 该函数用于解密文件夹, 代码如下:

```
void CFolderEncryDlg::OnUnEncry()
{
 CString strpath, despath;
 GetDlgItem(IDC_EDDIR) -> GetWindowText(strpath);
 int pos = strpath.Find(".");
 despath = strpath.Left(pos);
 if(despath.IsEmpty()) return;
 ::rename(strpath, despath);
}
```

### 举一反三

根据本实例, 读者可以:

- 将文件夹分类整理之后进行加密;
- 批量加密磁盘中的文件夹。

## 5.9 INI 文件

本节所讲述的是如何通过程序向 INI 文件中写入数据以及如何使用 INI 文件记录程序的配置信息。

### 实例 202 向 INI 文件中写入数据

这是一个可以提高分析能力的实例

实例位置: 光盘\mingrisoft\05\202

#### 实例说明

对于 INI 文件的读写操作都是通过系统所提供的 API 函数来实现的。本实例就是通过这些 API



函数向 INI 文件中写入不同的数据。如图 5.33 所示。

### 技术要点

在本实例中通过 3 个 API 函数实现向指定的 INI 文件中写入数据。这 3 个函数分别为 WritePrivateProfileSection、WritePrivateProfileString 和 WritePrivateProfileStruct。这 3 个函数分别实现了向 INI 文件中写入数据、字符串和结构。

### 实现过程

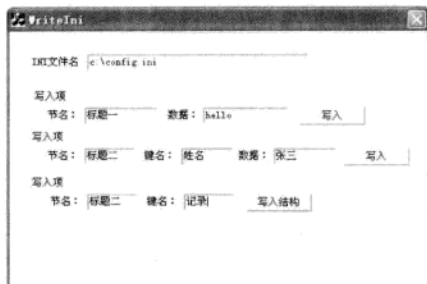


图 5.33 向 INI 文件中写入数据

(1) 新建名为 WriteIni 的对话框 MFC 工程。

(2) 在对话框上添加若干个静态文本控件用来标识需要输入数据的类别；添加若干个文本编辑控件用来输入向 INI 文件中写入的数据；添加 3 个按钮控件，用来向 INI 文件中写入 3 种不同的数据。

(3) 向 INI 文件中写入数据：

```
void CWriteIniDlg::OnButton1()
{
 CString filename,AppName,data;
 m_filename.GetWindowText(filename);
 m_edit2.GetWindowText(AppName);
 m_edit3.GetWindowText(data);
 WritePrivateProfileSection(AppName,data,filename);
}
```

(4) 向 INI 文件中写入字符串：

```
void CWriteIniDlg::OnButton2()
{
 CString filename,AppName,KeyName,data;
 m_filename.GetWindowText(filename);
 m_edit4.GetWindowText(AppName);
 m_edit6.GetWindowText(KeyName);
 m_edit5.GetWindowText(data);
 WritePrivateProfileString(AppName,KeyName,data,filename);
}
```

(5) 向 INI 文件中写入结构数据：

```
void CWriteIniDlg::OnButton3()
{
 CString filename,AppName,KeyName;
 m_filename.GetWindowText(filename);
 m_edit7.GetWindowText(AppName);
 m_edit8.GetWindowText(KeyName);
 Record record;
 record.ID = 1;
 ::memset((char *)record.Name,0,255);
 strcpy(record.Name,"123456");
 WritePrivateProfileStruct(AppName,KeyName,&record,sizeof(Record),filename);
}
```

### 举一反三

根据本实例，读者可以：

- 向 INI 文件中写入整型数据；
- 向 INI 文件中写入浮点型数据。

## 实例 203

### 使用 INI 文件保存配置信息

这是一个可以提高分析能力的实例

实例位置：光盘\mingrisoft\05\203

### 实例说明

在许多管理系统中 INI 文件用来记录程序运行时所需要的数据。这是因为 INI 文件符合信

息存储的最基本的格式“键名=键值”。并且在写入时还可以对数据进行分类,即“项”。程序运行如图 5.34 所示。

### 技术要点

该实例使用 `GetPrivateProfileStruct` 函数向 INI 文件中写入窗体运行时的标题信息和大小信息。在程序初始化时读取 INI 文件中的信息,程序关闭时将数据存储在 INI 文件中。结构的定义如下:

```
typedef struct _ConfigInfo
{
 int height;
 int width;
 int left;
 int top;
}ConfigInfo;
```



图 5.34 使用 INI 文件保存配置信息

### 实现过程

- (1) 新建名为 FolderEncry 的对话框 MFC 工程。
- (2) 在对话框中添加一个静态文本控件,设置标题为“设置标题名称”;添加一个文本编辑框控件,用来输入窗体的新标题名称;添加两个按钮控件,分别设置其标题为“确定”和“取消”。

- (3) 程序初始化时读取 INI 文件中的信息,代码如下:

```
m_ConfigPath = GetAppPath() + "Config.ini";
CString Caption;
::GetPrivateProfileString("配置信息","标题","",Caption.GetBuffer(0),
 255,m_ConfigPath);
if (Caption != "")
{
 SetWindowText(Caption);
}
ConfigInfo configinfo;
if (GetPrivateProfileStruct("配置信息","位置",&configinfo,
 sizeof(ConfigInfo),m_ConfigPath))
{
 CRect rect;
 rect.left = configinfo.left;
 rect.top = configinfo.top;
 rect.right = rect.left + configinfo.width;
 rect.bottom = rect.top + configinfo.height;
 this->MoveWindow(&rect,true);
}
```

- (4) 程序关闭时向 INI 文件中存储配置信息,代码如下:

```
void CConfigINIDlg::OnDestroy()
{
 CString caption;
 m_caption.GetWindowText(caption);
 if (caption != "")
 {
 ::WritePrivateProfileString("配置信息","标题",caption,m_ConfigPath);
 }
 ConfigInfo configinfo;
 CRect rect;
 this->GetWindowRect(&rect);
 configinfo.height = rect.Height();
 configinfo.width = rect.Width();
 configinfo.left = rect.left;
 configinfo.top = rect.top;
 ::WritePrivateProfileStruct("配置信息","位置",&configinfo,sizeof(ConfigInfo),m_ConfigPath);
 CDialog::OnDestroy();
}
```

### 举一反三

根据本实例,读者可以:

- 利用 INI 文件存款系统日志;
- 利用 INI 文件保存表格控件信息。

## 5.10 其 他

前面章节分别讲解了与文件操作相关的一些实例, 本节将再通过几个实例, 对文件管理方面的知识做一下补充。

### 实例 204 文件分割器

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\05\204

#### 实例说明

文件分割器主要针对移动存储设备容量小, 而要复制的文件又很大这种情况而开发的。通过文件分割器可以将文件分割成任意大小, 然后复制到小容量的移动存储设备中, 要使用文件时通过文件分割器将文件合并即可。运行程序, 单击“选择要分割的文件”按钮选择文件, 单击...按钮可以改变文件分割后的保存路径, 最后在相应的编辑框内设置分割的大小, 如图 5.35 所示, 单击“分割”按钮就可以实现文件的分割; 要合并文件需要通过“选择分割配置文件”按钮选择分割时生成的 INI 文件, 单击“合并”按钮, 就可以将文件合并。

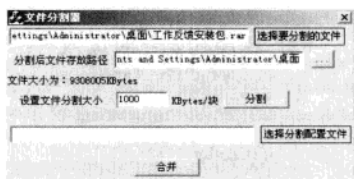



图 5.35 文件分割器

#### 技术要点

本实例主要通过对文件的读和写来实现文件分割的。首先通过指定的大小来分配空间, 再通过 CFile 类的 Read 方法实现读取要分割的文件到分配的空间中, 然后在循环 CreateFile 函数来创建文件, 再通过 Cfile 类的 Write 方法将读取出来的内容写入到新建的文件中, 读取完成后将文件的大小和分割后的块数等信息写入到 INI 文件中。分割后的各个子文件名都保持一定的规律性, 这个规律是文件名后加“part”加“序号”加“下划线”加“扩展名”加“.dat”。合并时读取 INI 文件, 然后在一个循环中通过文件的块数将各个子文件合并在一起。

 注意: 合并时 INI 文件要和分割后的文件保存在同一目录下。

#### 实现过程

- (1) 新建名为 FilePartition 的对话框 MFC 工程。
- (2) 在对话框上添加 4 个文本编辑控件, 设置 ID 属性分别是 IDC\_EDFILEPATH、IDC\_EDSAVEPATH、IDC\_EDSIZE 和 IDC\_EDINI; 添加 5 个按钮控件, 设置 ID 属性分别是 IDC\_BTADD、IDC\_BTDIR、IDC\_BTPartition、IDC\_BTSELPART 和 IDC\_BTOMBIN, 设置 Caption 属性分别为“选择要分割的文件”、“...”、“分割”、“选择分割配置文件”和“合并”。
- (3) “分割”按钮的实现函数, 该函数用于实现按指定大小分割文件, 并将分割结果保存成 INI 文件, 代码如下:

```
void CFilePartitionDlg::OnPartition()
{
 CFile *readfile,*writefile;
 DWORD filelen,readlen,poslen;
 CString name,path,desname;
 //获得将要分割的文件的全路径
 GetDlgItem(IDC_EDFILEPATH)->GetWindowText(name);
 //获得分割后文件的存放路径
 GetDlgItem(IDC_EDSAVEPATH)->GetWindowText(path);
 CString strsize;
 //获得文件分割块的大小
```

```
GetDlgItem(IDC_EDSIZE)->GetWindowText(strsize);
if(strsize.IsEmpty())return;

DWORD partsize=atoi(strsize)*1024;
BYTE *b=new BYTE[partsize];
readfile=new CFile(name,CFile::modeRead);
filelen=readfile->GetLength();
int i=1;
//在循环中根据文件的大小和用户设定的大小创建若干文件，并向文件中写入数据
while(1)
{
 ZeroMemory(b,partsize);
 desname.Format("%s\\%s%part%d_%.s.dat",path,filenamenoeext,i,filenameeeext);
 //创建文件块
 HANDLE hfile::CreateFile(desname,GENERIC_WRITE|GENERIC_WRITE,
 0,0,CREATE_NEW,FILE_ATTRIBUTE_NORMAL,0);
 CloseHandle(hfile);
 writefile=new CFile(desname,CFile::modeWrite);
 readlen=readfile->Read(b,partsize);
 poslen=readfile->GetPosition();
 writefile->Write(b,readlen);
 writefile->Close();
 i++;
 if(poslen==filelen)break;
}
readfile->Close();
delete writefile;
delete readfile;
//设置INI文件的路径
char buf[128];
::GetCurrentDirectory(128,buf);
CString inifile,pageend,size;
inifile.Format("%s\\%.s.ini",buf,filenamenoeext);
size.Format("%d",filelen);
pageend.Format("%d",i);
//将原来文件的信息及分割后的文件数写入到INI文件中
::WritePrivateProfileString("FilePartition","name",filenamenoeext,inifile);
::WritePrivateProfileString("FilePartition","ext",filenameeeext,inifile);
::WritePrivateProfileString("FilePartition","pageend",pageend,inifile);
::WritePrivateProfileString("FilePartition","size",size,inifile);
AfxMessageBox("分割成功");
}
```

(4) “选择分割配置文件”按钮的实现函数，该函数用于选择分割后生成的 INI 文件，代码如下：

```
void CFilePartitionDlg::OnSelectPart()
{
 CFileDialog log(TRUE,NULL,NULL,OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT|
 OFN_ALLOWMULTISELECT,"INI 文件(*.ini)*.inil",NULL);
 if(log.DoModal()==IDOK)
 {
 CString ininame=log.GetPathName();
 GetDlgItem(IDC_EDINI)->SetWindowText(ininame);
 }
}
```

(5) “合并”按钮的实现函数，该函数通过 INI 文件将文件合并，代码如下：

```
void CFilePartitionDlg::OnOmbin()
{
 CFile *readfile,*writefile;
 CString ininame,inidir;
 //获得分割文件时生成的INI文件
 GetDlgItem(IDC_EDINI)->GetWindowText(ininame);
 //获得INI文件所在目录
 inidir=FindPath(ininame);
 //获得分割前文件信息包括最后块序号，文件名（无扩展名），扩展名，块大小
 char pagenum[128],pagename[128],pageext[128],size[128];
 ::GetPrivateProfileString("FilePartition","name","",pagename,128,ininame);
 ::GetPrivateProfileString("FilePartition","ext","",pageext,128,ininame);
 ::GetPrivateProfileString("FilePartition","pageend","",pagenum,128,ininame);
 ::GetPrivateProfileString("FilePartition","size","",size,128,ininame);
 int pagecount=atoi(pagenum);
 CString desname; //合并后文件保存路径
 CString srcname; //合并的文件的路径
 desname.Format("%s\\%.s%.s",inidir,pagename,pageext);
 HANDLE hfile::CreateFile(desname,GENERIC_WRITE|GENERIC_WRITE,
 0,0,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,0);
 CloseHandle(hfile);
```



```

writefile=new CFile(desname,CFile::modeWrite);
for(int i=1;i<pagecount;i++)
{
 srcname.Format("%s\\%spart%d_%s.dat",inidir,pagename,i,pageext);
 readfile=new CFile(srcname,CFile::modeRead);
 DWORD filelen=readfile->GetLength();
 BYTE *b=new BYTE[filelen];
 readfile->Read(b,filelen);
 writefile->Write(b,filelen);
 readfile->Close();
 delete b;
}
writefile->Close();
delete writefile;
delete readfile;
AfxMessageBox("合并完成");
}

```

(6) “...”按钮的实现函数,该函数用于更改文件分割后的保存路径,代码如下:

```

void CFilePartitionDlg::OnChangeEdfilepath()
{
 CString desdirname,temp;
 GetDlgItem(IDC_EDFILEPATH)->GetWindowText(desdirname);
 int pos=desdirname.Find(filename);
 desdirname=desdirname.Left(pos-1);
 GetDlgItem(IDC_EDSAVEPATH)->SetWindowText(desdirname);
}

```

(7) 自定义函数 FindPath,实现通过全路径得到文件名,代码如下:

```

CString CFilePartitionDlg::FindPath(CString path)
{
 CString strpathname; //文件所在的路径
 CString strtemp; //保存全路径
 strtemp=path;
 int pos;
 int leftpos=0;
 pos=strtemp.Find("\\"); //查找子串
 while(pos!=-1)
 {
 leftpos+=pos;
 leftpos++;
 strtemp=strtemp.Right(strtemp.GetLength()-pos-1);
 pos=strtemp.Find("\\");
 }
 strpathname=path.Left(leftpos);
 return strpathname;
}

```

## 举一反三

根据本实例,读者可以:

- 多媒体文件的分割;
- 通过分割文件加密文件。

## 实例 205 用 WinRar 压缩和解压文件

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\05\205

## 实例说明

在网络中传输文件时,如果文件过大会影响文件传递的效果和速度。如果压缩后再传递,会提高传递速度并减少了时间。本实例将利用 WinRar 工具来压缩和解压缩文件。首先通过“浏览”按钮打开扩展名为 RAR 的文件,通过单击“开始解压”按钮将文件解压,解压后的文件和 RAR 文件放在统一目录下;通过“浏览”按钮选择将要压缩文件的文件夹,单击“开始压缩”按钮后,所选文件夹的全部文件都被压缩在 RAR 文件中。

## 技术要点

本实例通过 ShellExecute 函数调用 WinRar.exe 实现文件的压缩和解压。WinRar.exe 的语法如下。

解压:

WINRAR X [-switches] [Files] [@File lists] [destination folder]

压缩:

WINRAR A [-switches] [Files] [@File lists]

## 实现过程

(1) 新建名为 UseRar 对话框 MFC 工程。

(2) 在对话框上添加 2 个静态文本控件; 添加 2 个文本编辑控件, 设置 ID 属性分别为 IDC\_EDSOURCE 和 IDC\_EDDES, 添加 4 个按钮控件。

(3) “浏览”(解压)按钮的实现函数, 该函数用于添加需要解压的 RAR 文件, 代码如下:

```
void CUseRarDlg::OnAdd()
{
 CFileDialog file(true, "rar", NULL, OFN_HIDEREADONLY|OFN_FILEMUSTEXIST|
 OFN_EXPLORER, "RAR文件(*.rar)*.rar", this);
 if(file.DoModal()==IDOK)//显示文件打开对话框
 {
 CString path=file.GetPathName();
 GetDlgItem(IDC_EDSOURCE)->SetWindowText(path);
 }
}
```

(4) “开始解压”按钮的实现函数, 该函数用于调用 winrar.exe 对文件进行解压, 代码如下:

```
void CUseRarDlg::OnUnZip()
{
 CString source, path, temp;
 GetDlgItem(IDC_EDSOURCE)->GetWindowText(source);
 int pos=source.Find(".");
 path=source.Left(pos);
 temp.Format("X %s %s", source, path);
 ::CreateDirectory(path, NULL); //创建目录
 ::ShellExecute(NULL, "open", "WinRar.exe", temp, NULL, SW_SHOW);
}
```

(5) “浏览”(压缩)按钮的实现函数, 该函数用于设置将要压缩文件所在文件夹, 代码如下:

```
void CUseRarDlg::OnView()
{
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer, MAX_PATH);
 bi.hwndOwner=GetSafeHwnd();
 bi.pidlRoot=NULL;
 bi.pszDisplayName=buffer;
 bi.lpszTitle="选择一个文件夹";
 bi.ulFlags=BIF_EDITBOX;
 bi.lpfm=NULL;
 bi.lParam=0;
 bi.iImage=0;
 LPITEMIDLIST pList=NULL;
 if((pList=SHBrowseForFolder(&bi))!=NULL)
 {
 char path[MAX_PATH];
 ZeroMemory(path, MAX_PATH);
 SHGetPathFromIDList(pList, path);
 GetDlgItem(IDC_EDDES)->SetWindowText(path);
 }
}
```

(6) “压缩”按钮的实现函数, 该函数用于将指定文件夹下的文件压缩成 mingrisoft.rar 文件, 代码如下:

```
void CUseRarDlg::OnZip()
{
 CString des, temp, path, rarpath;
 GetDlgItem(IDC_EDDES)->GetWindowText(des);
 path.Format("%s*.%", des);
 rarpath.Format("%s\\mingrisoft.rar", des);
 temp.Format("a %s %s", rarpath, path);
}
```

```

::ShellExecute(NULL,"open","WinRar.exe",temp,NULL,SW_SHOW);//执行文件
}

```

## 举一反三

根据本实例，读者可以：

- 制作打包程序时压缩文件夹；
- 将文件压缩后存储到指定的文件夹当中。

## 实例 206 捆绑可执行文件

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\05\206

## 实例说明

捆绑应用程序属于一种隐藏式应用程序运行，该方法将一个或多个应用程序写入一个可执行文件中，当这个可执行文件运行时会根据判定条件运行该应用程序中的其他程序。以这种方式运行的程序隐密性强，可增加安全性。如图 5.36 所示。

## 技术要点

该实例的设计过程主要是利用文件的读写技术对多个文件进行合并操作。这些操作主要是利用二进制文件的打开、读和写操作，分别对应 fopen、fread 和 fwrite 等二进制文件操作函数。

在运行生成后的捆绑文件时，先从这个可执行文件中读取需要运行的可执行文件的数据存储到磁盘中，并生成临时的可执行文件。再通过 CreateProcess 函数调用临时可执行文件，CreateProcess 函数实现代码如下：

```

void CBindAppDlg::RunExeFile(LPCTSTR lpExeFile)
{
 HANDLE hProc; //进程句柄
 PROCESS_INFORMATION procInfo; //进程信息
 STARTUPINFO startInfo; //状态信息
 memset(&startInfo, 0, sizeof(STARTUPINFO));
 startInfo.cb = sizeof(STARTUPINFO);
 //创建新的进程并运行
 CreateProcess(lpExeFile, NULL, NULL, NULL, FALSE,
 NORMAL_PRIORITY_CLASS, NULL, NULL, &startInfo, &procInfo);
 hProc = procInfo.hProcess;
 WaitForSingleObject(hProc, INFINITE); //等待进程结束
 unlink(lpExeFile);
}

```

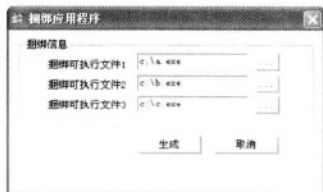


图 5.36 捆绑可执行文件

## 实现过程

- (1) 新建一个基于对话框的工程。
- (2) 在对话框上添加 3 个编辑框控件，分别用来显示捆绑的 2 个文件路径和所生成的捆绑文件的路径。添加 3 个按钮，用来设置文件路径。添加一个名为“生成”的按钮，用来执行文件捆绑的操作。

- (3) 首先定义一个结构用来记录可执行文件的长度和需要替换的文件长度。实现代码如下：

```
#define FINDFLAG 0x12345678
```

```
//定义一个结构，fileFlag为全局变量，用于记录可执行文件的长度
struct FILE_FLAG
```

```

{
 UINT nFindFlag; //查找标识
 UINT nFileLen; //文件长度
} fileFlag = {FINDFLAG, 0};

```

- (4) 在窗口中单击“生成”按钮，将在窗体中设置的前两个可执行文件中的数据与程序的

自身数据写入第3个可执行文件中,并根据可执行文件的自身大小替换 FINDFLAG 资源数据。  
实现代码如下:

```
//生成可执行文件
void CBindAppDlg::OnBuild()
{
 //读取文件1
 CString szFile1, szFile2, szFile3;
 m_File1.GetWindowText(szFile1);
 m_File2.GetWindowText(szFile2);
 m_File3.GetWindowText(szFile3);

 if (!szFile1.IsEmpty() && !szFile2.IsEmpty() && !szFile3.IsEmpty())
 {
 FILE* pflSelf = NULL; //可执行文件自身
 FILE* pflFinal = NULL; //最终生成的可执行文件
 FILE* pflFirst = NULL; //第一个捆绑的文件
 FILE* pflSecond = NULL; //第二个捆绑的文件
 struct _stat fileState;
 //获取自身的可执行文件信息
 _stat(m_szSelfName, &fileState);
 //获取文件长度
 DWORD dwFileLen = fileState.st_size;
 fileFlag.nFileLen = dwFileLen;
 //为文件分配缓冲区
 BYTE* pFileBuffer = (BYTE*) malloc(dwFileLen);
 //打开当前文件
 pflSelf = fopen(m_szSelfName, "rb");
 if (pflSelf == NULL)
 {
 MessageBox("打开自身文件失败!", "提示");
 return;
 }
 //读取文件
 DWORD dwRead = fread(pFileBuffer, sizeof(char), dwFileLen, pflSelf);
 fclose(pflSelf);
 if (dwRead != dwFileLen)
 {
 MessageBox("读取自身文件发生错误!", "提示");
 return;
 }

 //写入文件的长度
 UINT nFlag = FINDFLAG;
 //在可执行文件中查找FINDFLAG常量值,目的是定位到全局变量fileFlag的数据存储位置
 for(int i=0; i<dwFileLen-sizeof(UINT); i+=sizeof(UINT))
 {
 //在可执行文件中查找FINDFLAG即0x12345678,因为全局变量fileFlag中包含有0x12345678
 for(int j=0; j<sizeof(UINT); j++)
 {
 if (pFileBuffer[i+j] != ((BYTE*)&nFlag)[j])
 break;
 }
 if (j == sizeof(UINT)) //在可执行文件中发现了0x12345678数据
 {
 //修改可执行文件中fileFlag的数据存储位置上的数据
 memcpy(pFileBuffer+i, &fileFlag, sizeof(FILE_FLAG));
 break;
 }
 }
 if (i >= dwFileLen-sizeof(UINT)) //在可执行文件中没有发现0x12345678数据
 {
 free(pFileBuffer);
 MessageBox("定义可执行文件错误!", "提示");
 return;
 }

 //写入第一个可执行文件
 if (_stat(szFile1.GetBuffer(0), &fileState) != 0 || fileState.st_size == 0)
 {
 free(pFileBuffer);
 MessageBox("读取第一个绑定的文件失败!", "提示");
 return;
 }
 szFile1.ReleaseBuffer();
 //创建最终的可执行文件
 pflFinal = fopen(szFile3.GetBuffer(0), "wb");
```



```

if (pflFinal == NULL)
{
 free(pFileBuffer);
 MessageBox("创建可执行文件失败!", "提示");
 return;
}
szFile3.ReleaseBuffer();
//首先写入自身文件到最终的可执行文件中
fwrite(pFileBuffer, sizeof(char), dwFileLen, pflFinal);
//打开第一个可执行文件
pflFirst = fopen(szFile1.GetBuffer(0), "rb");
if (pflFirst == NULL)
{
 fclose(pflFinal);
 //删除生成的文件
 ...//
 free(pFileBuffer);
 MessageBox("打开第一个绑定的文件失败!", "提示");
 return;
}
szFile1.ReleaseBuffer();
//写入第一个要捆绑的文件长度
fwrite(&fileState.st_size, sizeof(char), sizeof(fileState.st_size), pflFinal);
//将第一个文件写入到最终的可执行文件中
free(pFileBuffer);
pFileBuffer = (BYTE*)malloc(fileState.st_size);
fread(pFileBuffer, sizeof(char), fileState.st_size, pflFirst);
fwrite(pFileBuffer, sizeof(char), fileState.st_size, pflFinal);
fclose(pflFirst);
free(pFileBuffer);
//写入第二个要捆绑的文件
if (_stat(szFile2.GetBuffer(0), &fileState) != 0 || fileState.st_size == 0)
{
 MessageBox("读取第二个绑定的文件失败!", "提示");
 return;
}
szFile2.ReleaseBuffer();
pflSecond = fopen(szFile2.GetBuffer(0), "rb");
if (pflSecond == NULL)
{
 MessageBox("打开第二个绑定的文件失败!", "提示");
 return;
}
szFile2.ReleaseBuffer();
pFileBuffer = (BYTE*)malloc(fileState.st_size);
fread(pFileBuffer, sizeof(char), fileState.st_size, pflFirst);
fwrite(pFileBuffer, sizeof(char), fileState.st_size, pflFinal);
fclose(pflSecond);
free(pFileBuffer);
MessageBox("绑定完成!", "提示");
}
}

```

(5) 在窗体类中实现 RunBindExe 方法, 这个方法是在已生成的捆绑文件运行时执行的。该方法先将存储在捆绑文件中的其他可执行文件数据读出, 并生成对应的可执行文件。实现代码如下:

```

//运行绑定的EXE文件
void CBindAppDlg::RunBindExe()
{
 FILE* pflSelf = NULL; //当前文件自身
 FILE* pflFirst = NULL; //捆绑的第一个文件
 FILE* pflSecond = NULL; //捆绑的第二个文件
 char pszFirstFile[] = "First.exe";
 char pszSecondFile[] = "Second.exe";

 pflSelf = fopen(m_szSelfName, "rb");
 if (pflSelf == NULL)
 {
 return;
 }
 pflFirst = fopen(pszFirstFile, "wb");
 if (pflFirst == NULL)
 {
 fclose(pflSelf);
 return;
 }
 //定位到第一个文件的结尾

```

```

fseek(pflSelf, fileFlag.nFileLen, SEEK_SET);
//读取第一个绑定的文件长度

UINT nFileLen = 0;
if (fread(&nFileLen, sizeof(char), sizeof(UINT), pflSelf) == 0)
{
 fclose(pflSelf);
 fclose(pflFirst);
 return;
}
//向绑定的第一个文件中写入数据
BYTE* pBuffer = (BYTE*) malloc(nFileLen);
fread(pBuffer, sizeof(char), nFileLen, pflSelf);
fwrite(pBuffer, sizeof(char), nFileLen, pflFirst);
fclose(pflFirst);

//向绑定的第二个文件中写入数据
pflSecond = fopen(pszSecondFile, "wb");
if (pflSecond == NULL)
{
 fclose(pflSelf);
 free(pBuffer);
 return;
}
int nReadSize = 0;
while(nReadSize = fread(pBuffer, sizeof(char), nFileLen, pflSelf))
{
 fwrite(pBuffer, sizeof(char), nReadSize, pflSecond);
}
fclose(pflSecond);
free(pBuffer);
fclose(pflSelf);

RunExeFile(pszFirstFile);
RunExeFile(pszSecondFile);
}

```

### 举一反三

根据本实例，读者可以：

- 向可执行文件中写数据；
- 提取可执行文件中的资源。

## 实例 207 读写 XML 文件

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\05\207

### 实例说明

XML 文件是可以自定义标签的文件，现在一般应用程序都用它来保存配置。在 Windows 系统中，提供了读写 XML 文件中数据的接口，本实例使用接口的动态链接库是 msxml.dll。如图 5.37 所示。

### 技术要点

在对 XML 文件进行读写操作前应在 stdafx.h 头文件中导入 msxml.dll 动态库。代码如下：

```

#import "msxml.dll"
using namespace MSXML;

```

通过 IXMLDOMDocumentPtr 接口指针创建 XML 文档实例，然后通过 IXMLDOMNodeListPtr 接口指针和 IXMLDOMNodePtr 接口指针完成 XML 节点的添加与读取操作。

### 实现过程

- (1) 新建一个名为 ReadWriteXML 的对话框应用程序。

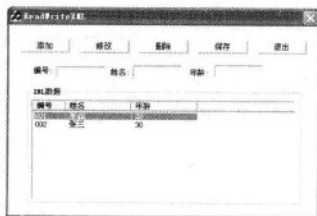


图 5.37 读写 XML 文件

(2) 向窗体中添加 5 个按钮控件, 分别设其标题为“添加”、“修改”、“删除”、“保存”和“退出”; 添加 3 个静态文本控件, 设置其标题为“编号”、“姓名”和“年龄”; 添加 1 个列表框视图控件。

(3) 读取 XML 文件中的数据到列表框控件中, 代码如下:

```
void CReadWriteXMLDlg::ReadXMLData()
{
 MSXML::IXMLDOMDocumentPtr xdoc;
 xdoc.CreateInstance(__uuidof(MSXML::DOMDocument)); //创建XML文档
 xdoc->load((_bstr_t)XMLFilePath); //载入XML文档
 IXMLDOMNodeListPtr nodelist=NULL; //节点列表
 IXMLDOMNodeListPtr rows=NULL;
 IXMLDOMNodeListPtr cols=NULL;
 nodelist=xdoc->selectNodes("XMLList"); //选择节点

 MSXML::IXMLDOMNodePtr Root; //根节点
 MSXML::IXMLDOMNodePtr Row;
 MSXML::IXMLDOMNodePtr Col;
 Root = nodelist->nextNode(); //获取跟节点
 _bstr_t bstrname=Root->nodeName; //节点名称
 rows = Root->childNodes; //子节点列表
 m_list.DeleteAllItems();
 for (int i = 0 ; i < rows->length ; i++)
 {
 Row = rows->nextNode(); //下一个节点
 cols = Row->childNodes; //节点列表
 _bstr_t rowname = Row->nodeName; //节点名称
 m_list.InsertItem(m_list.GetItemCount(), "");
 for (int j = 0 ; j < cols->length ; j++)
 {
 Col = cols->nextNode();
 IXMLDOMNodePtr node = Col->firstChild;
 _bstr_t value = node->nodeValue;
 m_list.SetItemText(i,j,value);
 }
 }
}
```

(4) 保存列表框中的数据到 XML 文件中, 代码如下:

```
void CReadWriteXMLDlg::UpdateXMLData()
{
 MSXML::IXMLDOMDocumentPtr xdoc;
 xdoc.CreateInstance(__uuidof(MSXML::DOMDocument)); //创建MXL文档
 MSXML::IXMLDOMElementPtr Root = xdoc->createElement("XMLList");
 MSXML::IXMLDOMElementPtr Row;
 MSXML::IXMLDOMElementPtr Col;
 xdoc->appendChild(Root); //添加子节点
 for (int i = 0 ; i < m_list.GetItemCount() ; i++)
 {
 Row = xdoc->createElement("Row");
 Root->appendChild(Row);
 for (int j = 0 ; j < m_list.GetHeaderCtrl()->GetItemCount() ; j++)
 {
 LVCOLUMN column;
 column.mask = LVCF_TEXT;
 column.cchTextMax = 255;
 char str[255];
 column.pszText = str;
 m_list.GetColumn(j,&column);

 Col = xdoc->createElement(column.pszText);
 Row->appendChild(Col);
 CString values;
 values = m_list.GetItemText(i,j);
 Col->appendChild(xdoc->createTextNode((_bstr_t)values));
 }
 }
 xdoc->save((_bstr_t)XMLFilePath);
}
```

### 举一反三

根据本实例, 读者可以:

- 使用 XML 文件写入程序配置信息;
- 从 XML 文件中读取程序的配置信息。

## 第 6 章

# 操作系统与Windows相关程序

- 启动相关
- 桌面相关
- 系统监控
- 线程同步
- 动态链接库
- 磁盘相关
- 系统相关
- 程序相关
- 鼠标、键盘相关

Visual C++



## 6.1 启动相关

本节中通过几个与系统启动有关的实例程序来系统地介绍与 Windows 系统启动相关的知识,其中包括进入 Windows XP 前发出警告、实现关机/重启计算机和将程序设置成为开机自动执行的程序等。

### 实例 208 进入 WinXP 前发出警告

这是一个自娱自乐的实例

实例位置: 光盘\mingrisoft\06\208

#### 实例说明

本实例实现了进入 Windows XP 系统前,显示一些提示、欢迎和警告类的信息。运行程序,在文本框中输入相应内容,如图 6.1 所示,单击“添加提示信息”按钮后,设置信息将写入注册表,重新启动计算机后,在进入 Windows XP 系统前就会显示相应的信息。如果想要删除这些信息,单击“去除提示信息”按钮后,重新启动计算机后即可。

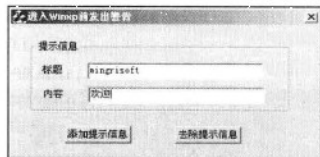


图 6.1 进入 Win XP 前发出警告

#### 技术要点

在程序中使用 RegCreateKey 等注册表函数在 Windows XP 系统注册表 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon 下面新建一个名为“Legalnoticecaption”的字符串值,将它的数据设置成提示信息的标题,如“警告”或“用户注意事项”等。在 Winlogon 次级主键下再新建一个名为“LegalNoticeText”的字符串值,输入提示信息,如“欢迎进入 Windows”等。这样,进入 Windows XP 前就会发出警告信息,弹出警告对话框信息。

#### 实现过程

(1) 新建一个名为 StartXP 的对话框 MFC 工程。

(2) 在对话框上添加两个静态文本控件,设置 Caption 属性分别为“标题”和内容;添加两个文本编辑控件,设置 ID 属性分别为 IDC\_EDITLE 和 IDC\_EDBODY;添加两个按钮控件,设置 ID 属性分别为 IDC\_BTADD 和 IDC\_BTMOVE,设置 Caption 属性分别为“添加提示信息”和“去除提示信息”。

(3) “IDC\_BTADD”按钮的实现函数,该函数用来在注册表中设置提示信息,代码如下:

```
void CStartXPDlg::OnAdd()
{
 CString strtitle, strbody;
 GetDlgItem(IDC_EDITLE)->GetWindowText(strtitle);
 GetDlgItem(IDC_EDBODY)->GetWindowText(strbody);
 HKEY sub;
 CString key="Software\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon";
 ::RegCreateKey(HKEY_LOCAL_MACHINE, key, &sub);
 RegSetValueEx(sub, "legalnoticecaption", NULL, REG_SZ, (BYTE*)strtitle.GetBuffer(0), strtitle.GetLength());
 RegSetValueEx(sub, "LegalNoticeText", NULL, REG_SZ, (BYTE*)strbody.GetBuffer(0), strbody.GetLength());
 RegCloseKey(sub);
}
```

(4) “IDC\_BTMOVE”按钮的实现函数,该函数用于清除提示信息,代码如下:

```
void CStartXPDlg::OnMove()
{
 HKEY sub;
 CString key="Software\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon";
 ::RegCreateKey(HKEY_LOCAL_MACHINE, key, &sub);
 ::RegDeleteValue(sub, "legalnoticecaption");
 ::RegDeleteValue(sub, "LegalNoticeText");
 RegCloseKey(sub);
}
```

## 举一反三

根据本实例，读者可以：

- 进入系统时弹出欢迎界面；
- 进入系统时显示计算机的所有者信息。

## 实例 209 实现关机、重启计算机

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\06\209

## 实例说明

在安装应用程序的过程中有时需要实现关闭或重启计算机的功能。本实例将介绍如何用 VC 控制计算机的关机和重启等。运行程序，如图 6.2 所示，单击窗体中的控制按钮，就可以重新启动、关闭或者注销计算机。

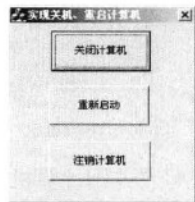


图 6.2 实现关机、重启计算机

## 技术要点

重新启动计算机、关闭计算机和注销计算机只需要 ExitWindowsEx 函数即可实现，但在 Windows 2000 系统下还需要使应用程序获得相关权限才可实现，这就需要 OpenProcessToken 函数、LookupPrivilegeValue 函数和 AdjustTokenPrivileges 函数的组合使用。

下面介绍这几个函数的用法。

OpenProcessToken 函数是获得应用程序权限的令牌，其语法如下：

```
BOOL OpenProcessToken(HANDLE ProcessHandle,DWORD DesiredAccess,PHANDLE TokenHandle);
```

参数说明：

- ProcessHandle：获得令牌的应用程序句柄。
- DesiredAccess：访问令牌的请求类型。
- TokenHandle：令牌句柄。

LookupPrivilegeValue 函数是获得权限的惟一标识值，语法如下：

```
BOOL LookupPrivilegeValue(LPCTSTR lpSystemName,LPCTSTR lpName,PLUID lpLuid);
```

参数说明：

- lpSystemName：权限所在的系统名称。
- lpName：权限的名称。
- lpLuid：权限的惟一标识。

AdjustTokenPrivileges 函数是设置令牌的权限，语法如下：

```
BOOL AdjustTokenPrivileges(HANDLE TokenHandle,BOOL DisableAllPrivileges,
 TOKEN_PRIVILEGES NewState,DWORD BufferLength,
 PTOKEN_PRIVILEGES PreviousState,PDWORD ReturnLength);
```

参数说明：

- TokenHandle：令牌句柄。
- DisableAllPrivileges：打开或关闭令牌权限。
- NewState：新的 TOKEN\_PRIVILEGES 结构对象，TOKEN\_PRIVILEGES 结构包含着一些权限信息及权限的属性。
- BufferLength：上一个 TOKEN\_PRIVILEGES 结构对象的缓存大小。
- PreviousState：上一个 TOKEN\_PRIVILEGES 结构对象。
- ReturnLength：返回上一个 TOKEN\_PRIVILEGES 结构对象所要求的缓存大小。

## 实现过程

- (1) 新建一个名为 ShutWindow 的对话框 MFC 工程。
- (2) 在对话框上添加 3 个按钮控件，设置 ID 属性分别为 IDC\_BTCCLOSE、IDC\_BTRESET 和 IDC\_BTLOGOUT，设置 Caption 属性分别为“关闭计算机”、“重新启动”和“注销计算机”。

- (3) 在 OnInitDialog 函数中设置实现关机的权限，代码如下：

```
BOOL CShutWindowDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 static HANDLE hToken;
 static TOKEN_PRIVILEGES tp;
 static LUID luid;
 OpenProcessToken(GetCurrentProcess(),TOKEN_ADJUST_PRIVILEGES|TOKEN_QUERY,
&hToken);
 LookupPrivilegeValue(NULL,SE_SHUTDOWN_NAME,&luid);
 tp.PrivilegeCount =1;
 tp.Privileges [0].Luid =luid;
 tp.Privileges [0].Attributes =SE_PRIVILEGE_ENABLED;
 AdjustTokenPrivileges(hToken,FALSE,&tp,sizeof(TOKEN_PRIVILEGES),NULL,NULL);

 return TRUE;
}
```

- (4) “关闭计算机”按钮实现函数，该函数用于关闭计算机，代码如下：

```
void CShutWindowDlg::OnClose()
{
 ExitWindowsEx(EWX_POWEROFF,0);
}
```

- (5) “重新启动”按钮实现函数，该函数用于重新启动计算机，代码如下：

```
void CShutWindowDlg::OnReset()
{
 ExitWindowsEx(EWX_REBOOT,0);
}
```

- (6) “注销计算机”按钮实现函数，该函数用于注销计算机，代码如下：

```
void CShutWindowDlg::OnLogout()
{
 ExitWindowsEx(EWX_LOGOFF,0);
}
```

## 举一反三

根据本实例，读者可以：

- 通过局域网控制关闭计算机；
- 远程控制计算机的重新启动和关闭。

### 实例 210

### 将程序设置成为开机自动执行的程序

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\06\210

## 实例说明

有些软件安装完成以后，在每次启动计算机时程序都会自动运行。本例讲解如何让程序开机自动运行。运行程序，如图 6.3 所示，选择程序中的复选框，单击“确定”按钮，相关信息会写入到注册表中，重新启动计算机后，本实例应用程序会在开机后自动运行。

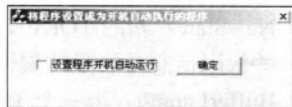


图 6.3 将程序设置成为开机自动执行的程序

## 技术要点

在程序中使用 RegCreateKey 等注册表函数在注册表 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run 下新建一个字符串, 值为要自动执行程序名, 将它的数据设置成程序所在目录, 即可将该程序设置自动执行。

## 实现过程

- (1) 新建一个名为 StartAutoRun 的对话框 MFC 工程。
- (2) 在对话框上添加复选框控件, 设置 ID 属性为 IDC\_CHSET, 设置程序开机自动运行, 添加成员变量 m\_chset; 添加一个按钮控件, 设置 ID 属性为 IDC\_BTENTER。
- (3) 主要程序代码如下:

```
void CStartAutoRunDlg::OnEnter()
{
 HKEY sub;
 char bufname[200];
 ::GetModuleFileName(NULL, bufname, 200);
 CString str;
 str.Format("%s", bufname);
 CString skey = "Software\\Microsoft\\Windows\\CurrentVersion\\Run";
 ::RegCreateKey(HKEY_LOCAL_MACHINE, skey, &sub);
 if(m_chset.GetCheck())
 {
 ::RegSetValueEx(sub, "StartAutoRun", NULL, REG_SZ,
 (const BYTE*)str.GetBuffer(0), str.GetLength());
 }
 else
 {
 ::RegDeleteValue(sub, "StartAutoRun");
 }
}
```

## 举一反三

根据本实例, 读者可以:

- 在应用程序首次启动的时候将其设置为开机自启动;
- 在安装程序时将应用程序设置为自启动。

## 6.2 磁盘相关

开发磁盘管理类软件在应用软件开发中占有很大的比重, 本节将通过几个典型实例介绍磁盘管理方面的相关技术。

### 实例 211 判断驱动器属性

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\06\211

#### 实例说明

驱动器的类型有很多, 如硬盘驱动器、软盘驱动器和光盘驱动器等, 本实例完成查看所选驱动器的类型的功能。运行程序, 在“驱动器列表”中选择相应的驱动器, 如 C:\, 驱动器的类型就会在下面显示出来, 如图 6.4 所示。

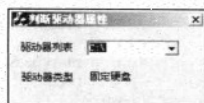


图 6.4 判断驱动器属性



## 技术要点

本实例主要通过 GetDriveType 函数来获得驱动器的类型, 它的语法如下:

```
UINT GetDriveType(LPCTSTR lpRootPathName);
```

参数说明:

- lpRootPathName: 驱动器盘符字符串。
- 函数的返回值可以判别驱动器类型, 该值如下。
  - DRIVE\_UNKNOWN: 不确定类型的驱动器。
  - DRIVE\_NO\_ROOT\_DIR: 不存在的驱动器。
  - DRIVE\_REMOVABLE: 可移动驱动器。
  - DRIVE\_FIXED: 硬盘驱动器。
  - DRIVE\_REMOTE: 远程驱动器。
  - DRIVE\_CDROM: 光盘驱动器。
  - DRIVE\_RAMDISK: RAM 磁盘驱动器。

## 实现过程

(1) 新建一个名为 DriverAttri 的对话框 MFC 工程。

(2) 在对话框上添加 3 个静态文本控件, 设置 ID 属性分别为 IDC\_DRIVELIST、IDC\_DRIVERTYPE 和 IDC\_TYPE, Caption 属性为“驱动器列表”和“驱动器类型”, 设置最后一个控件的 Caption 属性为空, 并添加成员变量 m\_type; 添加组合框控件, 设置 ID 属性为 IDC\_DRIVERCOMB, 添加成员变量 m\_drivercomb。

(3) 在 OnInitDialog 函数中初始化列表控件的内容, 代码如下:

```
BOOL CDriverAttriDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 size_t alldriver=::GetLogicalDriveStrings(0,NULL);
 _TCHAR *driverstr;
 driverstr=new _TCHAR[alldriver+sizeof(_T(""));
 if(GetLogicalDriveStrings(alldriver,driverstr)!=alldriver-1)//获取驱动器信息
 return FALSE;
 _TCHAR *pdriverstr=driverstr;
 size_t driversize=strlen(pdriverstr);
 while(driversize>0)
 {
 m_drivercomb.AddString(pdriverstr);
 pdriverstr+=driversize+1;
 driversize=strlen(pdriverstr);
 }
 return TRUE;
}
```

(4) 添加 IDC\_DRIVERCOMB 控件的 CBN\_SELCHANGE 实现函数, 该函数用于初始化组合框中的数据, 代码如下:

```
void CDriverAttriDlg::OnSelchangeDrivercomb()
{
 CString itemstr;
 int icursel=m_drivercomb.GetCurSel();
 m_drivercomb.GetLBText(icursel,itemstr);
 switch(::GetDriveType(itemstr))
 {
 case 2:
 m_type.SetWindowText("软驱");
 break;
 case 3:
 m_type.SetWindowText("固定硬盘");
 break;
 case 5:
 m_type.SetWindowText("光驱");
 break;
 case 4:
```

```

 m_type.SetWindowText("网络驱动器");
 break;
 case 6:
 m_type.SetWindowText("RAM");
 break;
 default:
 m_type.SetWindowText("未知");
 break;
 }
}

```

### 举一反三

根据本实例，读者可以：

- 在计算机连接新硬件（如移动硬盘）时检测该硬件。

## 实例 212 获取磁盘空间信息

本实例是一个与系统磁盘相关的软件

实例位置：光盘\mingrisoft\06\212

### 实例说明

在日常工作中，计算机磁盘空间的管理非常重要，例如在安装软件的时候，可以事先查看一下计算机中的磁盘空间大小，然后再根据剩余空间的大小决定将软件安装在哪个路径下。本实例演示的是一个显示系统中各个磁盘的使用情况的程序，程序运行之后，系统的磁盘驱动器盘符及驱动器的大小都显示在列表中，如图 6.5 所示。

### 技术要点

本实例主要通过 GetLogicalDriveStrings 函数来获得磁盘驱动器盘符，通过 GetDiskFreeSpaceEx 函数来获得驱动器空间。

### 实现过程

- 新建一个名为 DiskSpace 的对话框 MFC 工程。
- 在对话框上添加列表视图控件，设置 ID 属性为 IDC\_DISKLIST，添加成员变量 m\_disklist。
- 在工程中添加一个图标资源，设置 ID 属性为 IDI\_DISK。
- 主要程序代码如下：

```

BOOL CDiskSpaceDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_disklist.SetExtendedStyle(LVS_EX_GRIDLINES);
 m_disklist.InsertColumn(0,"磁盘驱动器",LVCFMT_LEFT,150);
 m_disklist.InsertColumn(1,"驱动器大小",LVCFMT_LEFT,150);
 imglist.Create(16,16,ILC_COLOR32|ILC_MASK,0,0);
 imglist.Add(::AfxGetApp()->LoadIcon(IDI_DISK));
 m_disklist.SetImageList(&imglist,LVSIL_SMALL);
 DWORD size;
 size=::GetLogicalDriveStrings(0,NULL);
 if(size!=0)
 {
 HANDLE heap=::GetProcessHeap();
 LPSTR lp=(LPSTR)HeapAlloc(heap,HEAP_ZERO_MEMORY,size*sizeof(TCHAR));
 ::GetLogicalDriveStrings(size*sizeof(TCHAR),lp);//获取驱动器信息
 while(*lp!=0)
 {
 m_disklist.InsertItem(0,lp,0);
 lp=_tcschr(lp,0)+1;
 }
 }
 ULARGE_INTEGER totalsize;
 ULARGE_INTEGER freesize;
 ULARGE_INTEGER availablesize;
 int num=m_disklist.GetItemCount();//获取磁盘数量

```

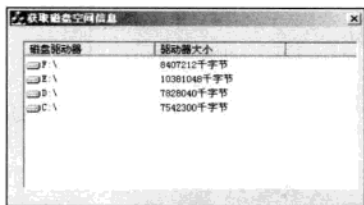


图 6.5 获取磁盘空间信息

```
for(int i=0;i<num;i++)
{
 CString str,temp;
 str=m_disklist.GetItemText(i,0);
 ::GetDiskFreeSpaceEx(str,&availablesize,&totalsize,&freesize);//获取磁盘空间
 temp.Format("%d千字节",totalsize.QuadPart/1024);
 m_disklist.SetItemText(i,1,temp);
}
return TRUE;
```

## 举一反三

根据本实例，读者可以：

- 开发磁盘空间报警程序。

## 实例 213 获取磁盘序列号

本实例是一个与系统磁盘相关的软件

实例位置：光盘\mingrisoft\06\213

## 实例说明

计算机的磁盘或硬盘格式化后都有对应的磁盘序列号。本例讲解的是如何获得磁盘的序列号。运行程序，系统的磁盘驱动器盘符和磁盘驱动器的序列号都会显示在列表中，如图 6.6 所示。

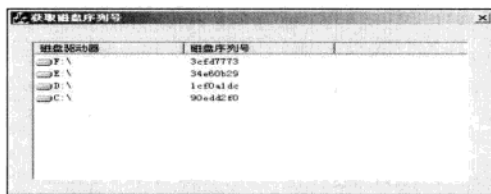


图 6.6 获取磁盘序列号

## 技术要点

本实例主要通过 GetVolumeInformation 函数来获取磁盘驱动器序列号，该函数是用来获得文件系统中根目录磁盘卷标信息的，其语法如下：

```
BOOL GetVolumeInformation(LPCTSTR lpRootPathName,LPTSTR lpVolumeNameBuffer,
 DWORD nVolumeNameSize,LPDWORD lpVolumeSerialNumber,
 LPDWORD lpMaximumComponentLength,LPDWORD lpFileSystemFlags,
 LPTSTR lpFileSystemNameBuffer,DWORD nFileSystemNameSize);
```

参数说明：

- lpRootPathName：根目录的名称。
- lpVolumeNameBuffer：存放卷标名称的缓存。
- nVolumeNameSize：卷标名称的大小。
- lpVolumeSerialNumber：卷标的序列号。
- lpMaximumComponentLength：最大文件名长度。
- lpFileSystemFlags：文件系统标志。
- lpFileSystemNameBuffer：存放文件系统名称的缓存。
- nFileSystemNameSize：存放文件系统名称的缓存的大小。

## 实现过程

- (1) 新建一个名为 DiskSerial 的对话框 MFC 工程。
- (2) 在对话上添加列表视图控件，设置 ID 属性为 IDC\_DISKLIST，添加成员变量 m\_disklist。
- (3) 在工程中添加图标资源，设置 ID 属性为 IDI\_DISK。
- (4) 主要程序代码如下：

```

BOOL CDiskSerialDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_disklist.SetExtendedStyle(LVS_EX_GRIDLINES);
 m_disklist.InsertColumn(0,"磁盘驱动器",LVCFMT_LEFT,150);
 m_disklist.InsertColumn(1,"磁盘序列号",LVCFMT_LEFT,150);
 imglist.Create(16,16,ILC_COLOR32,ILC_MASK,0,0);
 imglist.Add(::AfxGetApp()->LoadIcon(IDI_DISK));
 m_disklist.SetImageList(&imglist,LVSIL_SMALL);
 DWORD size;
 size=::GetLogicalDriveStrings(0,NULL);//获取驱动器信息
 if(size!=0)
 {
 HANDLE heap=::GetProcessHeap();
 LPSTR lp=(LPSTR)HeapAlloc(heap,HEAP_ZERO_MEMORY,size*sizeof(TCHAR));
 ::GetLogicalDriveStrings(size*sizeof(TCHAR),lp);
 while(*lp!=0)
 {
 m_disklist.InsertItem(0,lp,0);
 lp=_tcschr(lp,0)+1;
 }
 }
 LPCTSTR namebuf=new char[12];
 DWORD namesize=12;
 DWORD serialnumber;
 DWORD maxlen;
 DWORD fileflag;
 LPCTSTR sysnamebuf=new char[10];
 DWORD sysnamesize=10;
 int num=m_disklist.GetItemCount();//获取磁盘数量
 for(int i=0;i<num;i++)
 {
 CString str,temp;
 str=m_disklist.GetItemText(i,0);
 ::GetVolumeInformation(str,namebuf,namesize,&serialnumber,&maxlen,&fileflag,
 sysnamebuf,sysnamesize);
 temp.Format("%x",serialnumber);
 m_disklist.SetItemText(i,1,temp);
 }
 return TRUE;
}

```

### 举一反三

根据本实例，读者可以：

- 利用磁盘序列号加密软件。

## 实例 214 取消磁盘共享

本实例是一个与系统磁盘相关的软件

实例位置：光盘\mingrisoft\06\214

### 实例说明

在 Windows XP 系统或 Windows 2000 系统中，默认情况下，所有的磁盘都是共享的。如果用户想取消这些默认的共享需要在控制台输入命令来关闭，本实例将通过程序取消这些默认的共享设置，运行程序，在组合框中选择要取消共享的盘符，如图 6.7 所示，单击“关闭共享”按钮，便可实现取消磁盘共享的功能。

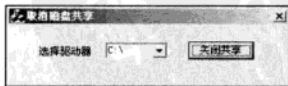


图 6.7 取消磁盘共享

### 技术要点

取消默认共享需要在控制台使用 NET 命令，例如要取消 C 盘的默认共享，需要使用 net share c:/del 命令。本实例主要通过 WinExec 函数来执行关闭磁盘默认共享的 NET 命令，该函数是在程序中执行其他可执行文件，语法如下：


UINT WinExec(LPCSTR lpCmdLine,UINT uCmdShow);

参数说明：

- lpCmdLine：命令行字符串，包括其它可执行文件所调用的参数。



- uCmdShow：执行其他可执行文件时窗体显示设置。

 注意：如果需要在启动计算机时取消默认共享，可以通过本实例介绍的方法编程取消本计算机所有默认共享，另外，如果将该执行程序放在启动文件夹或注册表指定位置中，即可实现开机取消默认共享。

## 实现过程

- (1) 新建一个名为 CloseShare 的对话框 MFC 工程。

(2) 在对话框上添加静态文本控件，设置 ID 属性为“选择驱动器”；添加组合框控件，设置 ID 属性为 IDC\_CMBDRIVER，添加成员变量 m\_drivercomb；添加按钮控件，设置 ID 属性为 IDC\_BTCLCLOSE，Caption 属性为“关闭共享”。

- (3) 在头文件 CloseShareDlg.h 中添加如下变量声明：

```
CString strsel;
```

(4) 在 OnInitDialog 函数中实现 IDC\_CMBDRIVER 控件中数据的初始化，将驱动器信息显示在组合框中，代码如下：

```
BOOL CCloseShareDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 DWORD size;
 size = ::GetLogicalDriveStrings(0, NULL);
 if (size != 0)
 {
 HANDLE heap = ::GetProcessHeap();
 LPSTR lp = (LPSTR)HeapAlloc(heap, HEAP_ZERO_MEMORY, size * sizeof(TCHAR));
 ::GetLogicalDriveStrings(size * sizeof(TCHAR), lp); // 获取驱动器信息
 while (*lp != 0)
 {
 m_drivercomb.AddString(lp);
 lp = _tcschr(lp, 0) + 1;
 }
 }
 return TRUE;
}
```

(5) 添加 IDC\_CMBDRIVER 控件的 CBN\_SELCHANGE 消息的实现函数，获得用户在组合框中选择的内容，代码如下：

```
void CCloseShareDlg::OnSelchangeCmbdriver()
{
 m_drivercomb.GetWindowText(strsel);
}
```

- (6) “取消共享”按钮的实现函数 OnCloseShare，实现对磁盘共享的取消，代码如下：

```
void CCloseShareDlg::OnCloseShare()
{
 CString strcmd;
 strcmd.Format("net.exe share %s /del", strsel);
 ::WinExec(strcmd, SW_HIDE); // 取消共享
}
```

## 举一反三

根据本实例，读者可以：

- 调用具有外部参数的应用程序；
- 自动获取磁盘信息并取消磁盘共享。

## 实例 215 格式化磁盘

本实例是一个与系统磁盘相关的软件

实例位置：光盘\mingrisoft\08\215

## 实例说明

磁盘是计算机存储数据的一种主要介质，磁盘主要分为两种：软盘（Floppy Disk）和硬盘

(Hard Disk)。硬盘在使用之前先进行低级格式化（也称物理格式化），然后进行分区，最后进行高级格式化（也称逻辑格式化），这样才能存储数据。而软盘不需要进行前两个步骤，直接进行高级格式化后就能够使用了。通常硬盘在出厂前已经进行了低级格式化，使用前直接分区和高级格式化后即可。高级格式化操作能将磁盘中的数据清空，并且能够恢复一些逻辑性的磁盘错误，因此会经常使用。运行本实例程序，通过组合框选择所要格式化的盘符，单击“格式化”按钮来格式化磁盘，运行程序如图 6.8 所示。

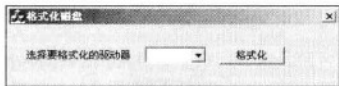


图 6.8 格式化磁盘

说明：本实例只是调用格式化磁盘的对话框，并没有真正去格式化。

## 技术要点

本实例通过调用 FormatDriver 函数显示格式化磁盘的对话框，该函数是 shell32.dll 文件中的函数，没有给出使用函数的头文件，因此需要使用调用动态链接库的方法实现对该函数的调用。具体方法：通过 LoadLibrary 函数来加载 shell32.dll 文件，通过 GetProcAddress 函数获得调用 FormatDriver 函数的指针，使用该指针可以实现对 FormatDriver 函数的调用。

下面介绍 FormatDriver 函数。语法：

FormatDriver (HWND hwnd, UINT drive,UINT fmtID,UINT options)

参数说明：

- hwnd：应用程序窗体句柄。
- drive：磁盘驱动器序号。
- fmtID：格式化 ID。
- options：格式化选项。

注意：格式化前请做好数据备份。

## 实现过程

- (1) 新建一个名为 FormatDriver 的对话框 MFC 工程。
- (2) 在对话框上添加静态文本控件，设置 ID 属性为“选择要格式化的驱动器”；添加组合框控件，设置 ID 属性为 IDC\_DRIVER，添加成员变量 m\_driver，添加按钮控件，设置 ID 属性为 IDC\_BTFORMAT，设置 Caption 属性为“格式化”。

- (3) 在头文件 FormatDriverDlg.h 添加如下变量声明：

```
int isel;
```

- (4) 在 OnInitDialog 函数中实现列表初始化，将驱动器信息显示在列表控件中，代码如下：

```
BOOL CFormatDriverDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 size_t alldriver::GetLogicalDriveStrings(0,NULL);
 _TCHAR *driverstr;
 driverstr=new _TCHAR[alldriver+sizeof(_T(""))];
 if(GetLogicalDriveStrings(alldriver,driverstr)!=alldriver-1)//获取驱动器信息
 return FALSE;
 _TCHAR *pdriverstr=driverstr;
 size_t driversize=strlen(pdriverstr);
 while(driversize>0)
 {
 m_driver.AddString(pdriverstr);
 pdriverstr+=driversize+1;
 driversize=strlen(pdriverstr);
 }
 isel=0;
 return TRUE;
}
```

- (5) 按钮“格式化”的实现函数，该函数用于调用格式化驱动器的对话框，具体代码如下：



```

void CFormatDriverDlg::OnFormat()
{
 typedef DWORD (WINAPI *MyFunc)(HWND hwnd,
 UINT drive,UINT fmtID,UINT options);
 HMODULE hModule=::LoadLibrary("shell32.dll");
 if(hModule)
 {
 MyFunc FormatDriver= (MyFunc) GetProcAddress(hModule, "SHFormatDrive");//获取格式化方法
 if(FormatDriver)
 FormatDriver(this->GetSafeHwnd(),isel,0xFFFF,0);//格式化驱动器
 }

 //keybd_event(VK_RETURN,NULL,NULL,NULL); //模拟按回车键
}

```

(6) 添加 IDC\_DRIVER 控件的 CBN\_SELCHANGE 消息实现函数，获得用户选择的驱动器信息，代码如下：

```

void CFormatDriverDlg::OnSelchangeDriver()
{
 CString strtemp;
 int i=m_driver.GetCurSel();
 m_driver.GetLBText(i,strtemp);
 char *chr;
 chr=new char[1];
 CString str;str=strtemp.Left(1);
 chr=str.GetBuffer(0);
 char ch=chr[0];
 isel=ch-'A';
}

```

### 举一反三

根据本实例，读者可以：

- 格式化软盘。

## 6.3 桌面相关

在 Windows 操作系统当中通常都是通过桌面来进行人机交互的，在本节将通过几个与桌面操作相关的程序来讲解控制桌面方面的相关技术。

### 实例 216 隐藏、显示开始按钮

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\06\216

### 实例说明

有些系统程序，为了不让用户使用开始菜单，可以隐藏任务栏上的“开始”按钮。本实例实现的是隐藏和显示开始按钮的功能。运行程序，如图 6.9 所示，单击“隐藏开始菜单”按钮，即可隐藏系统菜单的“开始”按钮，如图 6.10 所示；单击“显示开始菜单”按钮，即可显示系统菜单的“开始”按钮。

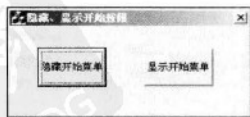


图 6.9 隐藏、显示开始按钮



图 6.10 隐藏“开始”按钮后的效果

## 技术要点

本实例主要通过使用 FindWindow 函数和 FindWindowEx 函数，查找具体窗体句柄，通过 ShowWindow 函数达到隐藏和显示窗体的目的。开始按钮在任务栏窗体上，任务栏的窗体名称是 Shell\_TrayWnd，这个名称可以通过 Visual C++ 开发环境自带的 SPY++ 工具获得。下面介绍 FindWindow 函数。

FindWindow 函数用于查找具体的窗体，语法如下：

HWND FindWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName);

参数说明：

- lpClassName：窗体类名称。
- lpWindowName：窗体名称。

## 实现过程

- (1) 新建一个名为 HideStartMenu 的对话框 MFC 工程。
- (2) 在对话框上添加两个按钮控件，设置 ID 属性分别为 IDC\_BTHIDE 和 IDC\_BTSHOW，设置 Caption 属性分别为“隐藏开始菜单”和“显示开始菜单”。
- (3) 按钮控件 IDC\_BTHIDE 的实现函数，该函数用于隐藏开始按钮，代码如下：

```
void CHideStartMenuDlg::OnHide()
{
 HWND parent = ::FindWindow("Shell_TrayWnd", "");
 HWND startmenu = ::FindWindowEx(parent, 0, "Button", NULL);
 if (startmenu != NULL)
 {
 ::ShowWindow(startmenu, SW_HIDE); // 隐藏窗口
 }
}
```

- (4) 按钮控件 IDC\_BTSHOW 的实现函数，该函数用于显示隐藏的开始按钮，代码如下：

```
void CHideStartMenuDlg::OnShow()
{
 HWND parent = ::FindWindow("Shell_TrayWnd", "");
 HWND startmenu = ::FindWindowEx(parent, 0, "Button", NULL);
 if (startmenu != NULL)
 {
 ::ShowWindow(startmenu, SW_SHOW); // 显示窗口
 }
}
```

## 举一反三

根据本实例，读者可以：

- 在开发触摸屏管理系统中禁止使用开始按钮。
- 在开发多媒体课件程序中禁止使用开始按钮。

## 实例 217 隐藏、显示桌面文件

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\06\217

### 实例说明

通常情况下，桌面上包含一些应用程序的快捷方式，看起来特别乱。本例可以实现将桌面隐藏或显示的功能。运行程序，单击“隐藏桌面文件”，即可将桌面上的快捷方式图标全部隐藏，单击“显示桌面文件”即可恢复桌面到原来的状态。程序运行如图 6.11 所示。

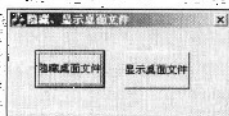


图 6.11 隐藏、显示桌面文件



## 技术要点

本实例主要通过使用 FindWindow 函数和 FindWindowEx 函数查找具体窗体句柄, 通过 ShowWindow 函数达到隐藏和显示窗体的目的。桌面的窗体名称是 Program Manager, 窗体类名称是 Progman。在 FindWindow 函数和 FindWindowEx 函数中可以单独使用窗体名称和窗体类名称。

## 实现过程

(1) 新建一个名为 HideDesktop 的对话框 MFC 工程。

(2) 在对话框上添加两个按钮控件, 设置 ID 属性分别为 IDC\_BTHIDE 和 IDC\_BTSHOW, 设置 Caption 属性分别为“隐藏桌面文件”和“显示桌面文件”。

(3) “隐藏桌面文件”按钮的实现函数, 该函数用于隐藏桌面, 代码如下:

```
void CHideDesktopDlg::OnHide()
{
 HWND desktop = ::FindWindow(NULL, "Program Manager");
 if (desktop != NULL)
 {
 ::ShowWindow(desktop, SW_HIDE);
 }
}
```

(4) “显示桌面文件”按钮的实现函数, 该函数用于显示隐藏的桌面, 代码如下:

```
void CHideDesktopDlg::OnShow()
{
 HWND desktop = ::FindWindow("Progman", NULL);
 if (desktop != NULL)
 {
 ::ShowWindow(desktop, SW_SHOW);
 }
}
```

## 举一反三

根据本实例, 读者可以开发:

- 在进行多媒体演示的过程中隐藏桌面上的文件;
- 当触摸屏程序运行的时候隐藏桌面上的文件。

## 实例 218

## 隐藏、显示 Windows 任务栏

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\06\218

## 实例说明

为了不让用户使用任务栏, 可以将任务栏隐藏起来。本例将讲解如何隐藏 Windows 任务栏。运行程序, 如图 6.12 所示, 单击“隐藏任务栏”按钮, Windows 任务栏将立即被隐藏; 单击“显示任务栏”按钮, Windows 任务栏将重新显示。

## 技术要点

本实例主要通过使用 FindWindow 函数和 FindWindowEx 函数查找具体窗体句柄, 通过 SetWindowPos 函数达到隐藏和显示窗体的目的。任务栏的窗体名称是 Shell\_TrayWnd, FindWindow 函数通过查找 Shell\_TrayWnd 可以获得任务栏窗体句柄, 再通过 SetWindowPos 函数控制任务栏的显示。SetWindowPos 函数主要是用来设置窗体显示位置的, 其语法如下:

```
BOOL SetWindowPos(HWND hWnd, HWND hWndInsertAfter, int x, int y, int cx, int cy, UINT nFlags);
```

参数说明:

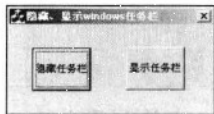


图 6.12 隐藏、显示 windows 任务栏

- hWnd: 应用程序窗体句柄。
- hWndInsertAfter: 窗体在 z 轴上的位置。取值如下:
- HWND\_BOTTOM: 窗体在底层。
- HWND\_NOTOPMOST: 窗体不在最顶层。
- HWND\_TOP: 窗体在顶层。
- HWND\_TOPMOST: 窗体总在顶层。
- x: 窗体的左顶点的 x 轴位置。
- y: 窗体的左顶点的 y 轴位置。
- cx: 窗体的宽度。
- cy: 窗体的高度。
- nFlags: 窗体的大小和位置标识, 主要取值如下。
  - SWP\_NOSIZE: 窗体大小不能改变。
  - SWP\_SHOWWINDOW: 窗体可以显示。
  - SWP\_HIDEWINDOW: 隐藏窗体。
  - SWP\_NOACTIVATE: 窗体处于非活动状态。

## 实现过程

- (1) 新建一个名为 HideTrayWnd 的对话框 MFC 工程。
- (2) 在对话框上添加两个按钮控件, 设置 ID 属性分别为 IDC\_BTHIDE 和 IDC\_BTSHOW, 设置 Caption 属性分别为“隐藏任务栏”和“显示任务栏”。
- (3) “隐藏任务栏”按钮的实现函数, 该函数用于隐藏任务栏, 代码如下:

```
void CHideTrayWndDlg::OnHide()
{
 HWND tray=::FindWindow("Shell_traywnd",NULL);
 if(tray!=NULL)
 {
 ::SetWindowPos(tray,HWND_TOPMOST,0,0,0,0,SWP_HIDEWINDOW);
 }
}
```

- (4) “显示任务栏”按钮的实现函数, 该函数用于将隐藏的任务栏显示, 代码如下:

```
void CHideTrayWndDlg::OnShow()
{
 HWND tray=::FindWindow("Shell_traywnd",NULL);
 if(tray!=NULL)
 {
 ::SetWindowPos(tray,HWND_TOPMOST,0,0,0,0,SWP_SHOWWINDOW);
 }
}
```

## 举一反三

根据本实例, 读者可以:

- 在开发软件安装程序时隐藏任务栏;
- 在演示多媒体程序时隐藏任务栏。

## 实例 219 随机修改系统桌面背景

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\06\219

## 实例说明

我们经常会为系统桌面更换漂亮的图片, 但有时许多图片都很喜欢, 所以就不知应更换哪一张图片。本实例实现了一个随机修改系统桌面背景的程序, 可以让使用者在图片文件列表中随机

选取一张图片作为系统的桌面背景。如图 6.13 所示。

## 技术要点

本实例是通过系统 API 函数 `SystemParametersInfo` 来完成系统桌面的更换。在程序中使用 `rand` 函数完成列表框中图片路径的随机选择。

## 实现过程

(1) 新建一个名为 `ChangeDesktop` 的对话框 MFC 工程。

(2) 在对话框中添加 1 个静态文本控件，设置标题为“图片路径”；添加 1 个文本编辑框控件，用来设置图片路径；添加 4 个按钮控件，分别设置其 `Caption` 属性为“导入”、“修改”、“随机修改”和“取消”；添加 1 个列表框控件。

(3) 导入指定目录下的位图图片，代码如下：

```
void CChangeDesktopDlg::OnLoad()
{
 CString strFilter = "*.bmp|.bmp|";
 CFileDialog filedialog(TRUE, NULL, NULL,
 OFN_FILEMUSTEXIST |
 OFN_HIDEREADONLY |
 OFN_PATHMUSTEXIST |
 OFN_ALLOWMULTISELECT,
 strFilter); // 定义文件浏览窗口

 if (filedialog.DoModal() == IDOK) // 显示文件浏览窗口
 {
 POSITION pos = filedialog.GetStartPosition();
 CString path, exname;
 while (NULL != pos)
 {
 path = filedialog.GetNextPathName(pos);
 exname = path.Right(4);
 exname.MakeUpper();
 if (exname == ".BMP")
 m_list.AddString(path);
 }
 path.MakeReverse();
 int p = path.FindOneOf("\\");
 path.MakeReverse();

 m_edit.SetWindowText(path.Left(path.GetLength()-p));
 }
}
```

(4) 将当前选择的图片作为系统桌面背景，代码如下：

```
void CChangeDesktopDlg::OnOK()
{
 CString path;
 if (m_list.GetCurSel() > 0)
 {
 m_list.GetText(m_list.GetCurSel(), path);
 // 更改系统桌面
 SystemParametersInfo(SPI_SETDESKWALLPAPER, 0, path.GetBuffer(0),
 SPIF_UPDATEINIFILE);
 }
}
```

(5) 随机选取图片作为系统桌面背景，代码如下：

```
void CChangeDesktopDlg::Onrand()
{
 int n = m_list.GetCount();
 int i = rand() % n;
 CString path;
 if (i > 0)
 {

```

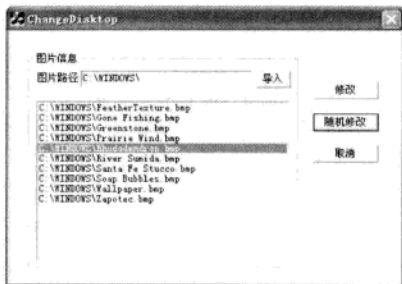


图 6.13 随机修改系统桌面背景

```
m_list.GetText(i,path);
//更改系统桌面
SystemParametersInfo(SPI_SETDESKWALLPAPER, 0, path.GetBuffer(0),
SPIF_UPDATEINIFILE);
m_list.SetCurSel(i);
}
}
```

## 举一反三

根据本实例，读者可以：

- 定时修改系统桌面。

## 实例 220 抓取桌面

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\06\220

## 实例说明

用户在日常的工作中，有时需要将电脑屏幕保存为图片，可是使用键盘上的<PrtScSysRq>键却不能抓取屏幕上的鼠标，本实例通过 Visual C++实现了抓取屏幕的功能，并且允许用户在抓图的过程中决定是否抓取鼠标。运行本实例，单击“...”按钮，在弹出的“浏览文件夹”对话框中选择图片存储路径，并选择是否抓取鼠标，设置抓屏热键，单击“保存”按钮保存设置，在需要抓图时直接按下设置的热键就可以将抓取的图片保存到设置的存储路径下。程序运行如图 6.14 所示。

## 技术要点

本示例中实现屏幕截图功能时，主要用 RegisterHotKey 函数、UnregisterHotKey 函数、GetDesktopWindow 函数和 CFile 类，下面对本实例中用到的关键技术进行详细讲解。

(1) RegisterHotKey 函数。RegisterHotKey 函数用于注册系统热键，该函数的语法格式如下：

BOOL RegisterHotKey( HWND hWnd, int id, UINT fsModifiers, UINT vk );

- hWnd：接收热键产生 WM\_HOTKEY 消息的窗口句柄。若该参数 NULL，传递给调用线程的 WM\_HOTKEY 消息必须在消息循环中进行处理。
- id：定义热键的标识符。
- fsModifiers：定义为了产生 WM\_HOTKEY 消息而必须与由 nVirtKey 参数定义的键一起按下的键。
- vk：定义热键的虚拟键码。

本实例中用于注册系统热键的代码如下：

```
m_HotKey.SetHotKey(wvk,wmod);
BOOL result = RegisterHotKey(this->GetSafeHwnd(),HOTKEY_GRASP,wmod,wvk);
if(!result)
{
 MessageBox("注册热键失败");
}
```

```
//设置热键控件中的键值
//注册系统热键
//判断是否注册成功
```

(2) UnregisterHotKey 函数。UnregisterHotKey 函数用于释放调用线程先前登记的热键，其语法格式如下：

BOOL UnregisterHotKey( HWND hWnd, int id );

- hWnd：与被释放的热键相关的窗口句柄。若热键不与窗口相关，则该参数为 NULL。
- id：定义被释放的热键的标识符。

(3) GetDesktopWindow 函数。GetDesktopWindow 函数用于获取系统桌面窗口，返回值是系统桌面窗口指针，通过该指针可以操作 Windows 桌面，其语法格式如下：

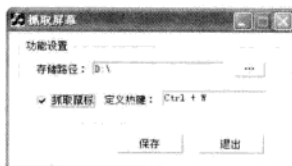


图 6.14 抓取桌面



static CWnd\* PASCAL GetDesktopWindow();

(4) CFile 类的 Open 方法。Open 方法用于打开一个文件,它可以和没有参数的构造函数 CFile 一起使用,用来以安全的方式打开文件,其语法格式如下:

virtual BOOL Open(LPCTSTR lpszFileName, UINT nOpenFlags, CFileException\* pError = NULL);

- lpszFileName: 将要打开的文件的路径,可以是绝对路径和相对路径,网络路径也可以。
- nOpenFlags: 一个定义了文件的共享和访问模式的 UINT。它指定了打开文件后的动作,可以用 OR (|) 操作符将选项组合起来,至少应有一个访问权限和一个共享选项,modeCreate 和 modeNoInherit 模式是可选的。
- pError: 一个异常的指针,一般情况下可以使用 NULL 指针,这个指针在打开文件过程中如果产生错误,Open 将抛出一个 CFileException 异常,而不是返回 FALSE。

(5) CFile 类的 WriteHuge 方法。WriteHuge 方法用于将缓冲区中的数据写入到文件,主要用来写入比较大的数据,其语法格式如下:

void WriteHuge(const void\* lpBuf, DWORD dwCount);

- lpBuf: 提供将要写入的数据的缓冲区。
- dwCount: 将要写入数据的大小。

(6) CFile 类的 Close 方法。Close 方法用于关闭 CFile 对象,其语法格式如下:

virtual void Close();

本实例中使用 CFile 类创建并保存文件的代码如下:

```
CFile file; //构造CFile对象
if(file.Open(name,CFile::modeCreate|CFile::modeWrite)) //创建位图文件
{
 file.WriteHuge(&bfb,sizeof(BITMAPFILEHEADER)); //写入位图文件头
 file.WriteHuge(pBInfo,sizeof(BITMAPINFOHEADER)); //写入文件头
 file.WriteHuge(lpData,size); //写入数据
 file.Close(); //关闭文件
}
```

## 实现过程

(1) 新建一个基于对话框的应用程序,将其窗体标题改为“屏幕截图工具”,勾选 Minimize box 属性,使对话框具有最小化按钮。

(2) 向对话框中添加 1 个群组控件、2 个静态文本控件、1 个编辑框控件、1 个复选框控件、1 个热键控件和 3 个按钮控件。

(3) 引用 windowsx.h 头文件和 math.h 头文件并声明常量,代码如下:

```
#include <windowsx.h>
#include "math.h"
#define HOTKEY_GRASP 10000
```

(4) 在主窗体初始化时读取 INI 文件中数据,根据读取的数据设置控件默认显示,并注册系统热键,代码如下:

```
m_Num = 1;
WORD wvk,wmod;
char bufwvk[8],bufwmod[8],bufpath[256],bufmouse[2];
GetPrivateProfileString("Set","path","",bufpath,256,"./System.ini"); //获得存储路径
m_Path.SetWindowText(bufpath); //显示存储路径
GetPrivateProfileString("Set","wvk","",bufwvk,8,"./System.ini"); //热键的虚拟代码
wvk = atoi(bufwvk); //转换为整数
GetPrivateProfileString("Set","wmod","",bufwmod,8,"./System.ini"); //获得修正符标志
wmod = atoi(bufwmod); //转换为整数
GetPrivateProfileString("Set","capmouse","",bufmouse,2,"./System.ini"); //获得是否捕捉鼠标标识
m_Cursor.SetCheck(atoi(bufmouse)); //设置复选框默认值
m_HotKey.SetHotKey(wvk,wmod); //设置热键控件默认值
BOOL result = RegisterHotKey(this->GetSafeHwnd(),HOTKEY_GRASP,wmod,wvk); //注册系统热键
if(!result) //判断热键是否注册成功
{
 MessageBox("注册热键失败");
}
```

(5) 处理“...”按钮的单击事件,在该事件中调用文件浏览对话框设置图片的存储路径,其实现代码如下:

```
void CGraspBmpDlg::OnButpath()
{
 // TODO: Add your control notification handler code here
 CString ReturnPach;
 TCHAR szPath[_MAX_PATH];
 BROWSEINFO bi;
 bi.hwndOwner = NULL;
 bi.pidlRoot = NULL;
 bi.lpszTitle = _T("文件浏览对话框");
 bi.pszDisplayName = szPath;
 bi.ulFlags = BIF_RETURNONLYFSDIRS;
 bi.lpfn = NULL;
 bi.lParam = NULL;
 LPITEMIDLIST pltemIDList = SHBrowseForFolder(&bi);
 if(pltemIDList)
 {
 if(SHGetPathFromIDList(pltemIDList,szPath))
 ReturnPach = szPath;
 }
 else
 {
 ReturnPach = "";
 }
 m_Path.SetWindowText(ReturnPach);
}
```

//字符串变量  
//保存路径变量  
// BROWSEINFO结构变量  
//HWND句柄  
//默认值为NULL  
//对话框标题  
//选择文件夹路径  
//标记  
//默认值为NULL  
//回调消息  
//显示文件浏览对话框  
//判断是否获得文件夹路径  
//获得文件夹路径  
//文件夹路径为空  
//显示存储路径

(6) 手动添加 WM\_HOTKEY 消息的处理函数, 在该函数中调用 StartGrasp 函数抓取屏幕并存成 BMP 位图文件, 实现代码如下:

```
void CGraspBmpDlg::OnHotKey(WPARAM wParam,LPARAM lParam)
{
 if(HOTKEY_GRASP == (int)wParam)
 {
 StartGrasp();
 }
}
```

//抓图

(7) 添加自定义函数 StartGrasp, 该函数用于进行屏幕抓图, 其实现代码如下:

```
void CGraspBmpDlg::StartGrasp()
{
 CString path;
 m_Path.GetWindowText(path);
 if(path.IsEmpty())
 {
 MessageBox("请选择文件存储位置");
 return;
 }
 CDC* pDeskDC = GetDesktopWindow()->GetDC();
 CRect rc;
 GetDesktopWindow()->GetClientRect(rc);
 CDC memDC;
 memDC.CreateCompatibleDC(pDeskDC);
 CBitmap bmp;
 bmp.CreateCompatibleBitmap(pDeskDC,rc.Width(),rc.Height());
 memDC.SelectObject(&bmp);
 BITMAP bitmap;
 bmp.GetBitmap(&bitmap);
 memDC.BitBlt(0,0,bitmap.bmWidth,bitmap.bmHeight,pDeskDC,0,0,SRCCOPY);
 if(m_Cursor.GetCheck())
 {
 CPoint point;
 GetCursorPos(&point);
 HICON hicon = (HICON)GetCursor();
 memDC.DrawIcon(point.x-10,point.y-10,hicon);
 }
 DWORD size=bitmap.bmWidthBytes*bitmap.bmHeight;
 LPSTR lpData=(LPSTR)GlobalAllocPtr(GPTR,size);
 int panelsize = 0;
 if(bitmap.bmBitsPixel<16)
 panelsize = pow(2,bitmap.bmBitsPixel*sizeof(RGBQUAD));
 BITMAPINFOHEADER *pBInfo = (BITMAPINFOHEADER*)LocalAlloc(LPTR,
 sizeof(BITMAPINFO)+panelsize);
 pBInfo->biBitCount = bitmap.bmBitsPixel;
 pBInfo->biClrImportant = 0;
 pBInfo->biCompression = 0;
 pBInfo->biHeight = bitmap.bmHeight;
 pBInfo->biPlanes = bitmap.bmPlanes;
 pBInfo->biSize = sizeof(BITMAPINFO);
}
```

//获得文件路径  
//判断文件路径是否为空  
//获取桌面画布对象  
//获取屏幕的客户区域  
//定义一个内存画布  
//创建一个兼容的画布  
//创建兼容位图  
//选中位图对象  
//获得图片信息  
//绘制图片  
//选择抓取鼠标  
//鼠标位置  
//获得鼠标图标句柄  
//绘制鼠标图标  
//图片数据大小  
//记录调色板大小  
//判断是否为真彩色位图  
//位图头指针  
//位图像素  
//位图高



```
pBInfo->biSizeImage = bitmap.bmWidthBytes*bitmap.bmHeight; //数据
pBInfo->biWidth = bitmap.bmWidth; //位图宽
pBInfo->biXPelsPerMeter = 0;
pBInfo->biYPelsPerMeter = 0;
GetDIBits(memDC.m_hDC,bmp,0,pBInfo->biHeight,lpData,
 (BITMAPINFO*)pBInfo,DIB_RGB_COLORS);
CString name,str;
CTime time = CTime::GetCurrentTime(); //抓图时间
str.Format("%04d",m_Num++);
if(path.Right(1) == "\\")
 name = path+time.Format("%Y%m%d")+str+".bmp"; //设置文件名及路径
else
{
 ReturnPath = ""; //文件夹路径为空
}
m_Path.SetWindowText(ReturnPath); //显示存储路径
```

(8) 手动添加 WM\_HOTKEY 消息的处理函数, 在该函数中调用 StartGrasp 函数抓取屏幕并生成 BMP 位图文件, 实现代码如下:

```
void CGraspBmpDlg::OnHotKey(WPARAM wParam,LPARAM lParam)
{
 if(HOTKEY_GRASP == (int)wParam)
 {
 StartGrasp(); //抓图
 }
}
```

(9) 添加自定义函数 StartGrasp, 该函数用于进行屏幕抓图, 其实现代码如下:

```
void CGraspBmpDlg::StartGrasp()
{
 CString path;
 m_Path.GetWindowText(path); //获得文件路径
 if(path.IsEmpty()) //判断文件路径是否为空
 {
 MessageBox("请选择文件存储位置");
 return;
 }
 CDC* pDeskDC = GetDesktopWindow()->GetDC(); //获取桌面画布对象
 CRect rc;
 GetDesktopWindow()->GetClientRect(rc); //获取屏幕的客户区域
 CDC memDC; //定义一个内存画布
 memDC.CreateCompatibleDC(pDeskDC); //创建一个兼容的画布
 CBitmap bmp;
 bmp.CreateCompatibleBitmap(pDeskDC,rc.Width(),rc.Height()); //创建兼容位图
 memDC.SelectObject(&bmp); //选中位图对象
 BITMAP bitmap;
 bmp.GetBitmap(&bitmap); //获得图片信息
 memDC.BitBlt(0,0,bitmap.bmWidth,bitmap.bmHeight,pDeskDC,0,0,SRCCOPY); //绘制图片
 if(m_Cursor.GetCheck()) //选择抓取鼠标
 {
 CPoint point;
 GetCursorPos(&point); //鼠标位置
 HICON hicon = (HICON)GetCursor(); //获得鼠标图标句柄
 memDC.DrawIcon(point.x-10,point.y-10,hicon); //绘制鼠标图标
 }
 DWORD size=bitmap.bmWidthBytes*bitmap.bmHeight; //图片数据大小
 LPSTR lpData=(LPSTR)GlobalAllocPtr(GPTR,size);
 int panelsize = 0; //记录调色板大小
 if(bitmap.bmBitsPixel<16) //判断是否为真彩色位图
 {
 panelsize = pow(2,bitmap.bmBitsPixel*sizeof(RGBQUAD));
 BITMAPINFOHEADER *pBInfo = (BITMAPINFOHEADER*)LocalAlloc(LPTR,
 sizeof(BITMAPINFO)+panelsize); //位图头指针
 pBInfo->biBitCount = bitmap.bmBitsPixel; //位图像素
 pBInfo->biClrImportant = 0;
 pBInfo->biCompression = 0;
 pBInfo->biHeight = bitmap.bmHeight; //位图高
 pBInfo->biPlanes = bitmap.bmPlanes;
 pBInfo->biSize = sizeof(BITMAPINFO);
 pBInfo->biSizeImage = bitmap.bmWidthBytes*bitmap.bmHeight; //数据
 pBInfo->biWidth = bitmap.bmWidth; //位图宽
 pBInfo->biXPelsPerMeter = 0;
 pBInfo->biYPelsPerMeter = 0;
 GetDIBits(memDC.m_hDC,bmp,0,pBInfo->biHeight,lpData,
 (BITMAPINFO*)pBInfo,DIB_RGB_COLORS);
 CString name,str;
 CTime time = CTime::GetCurrentTime(); //抓图时间
```



```
str.Format("%04d",m_Num++);
if(path.Right(1) == "\\")
 name = path+time.Format("%Y%m%d")+str+".bmp"; //设置文件名及路径
else
 name = path+"\\."+time.Format("%Y%m%d")+str+".bmp"; //设置文件名及路径
BITMAPFILEHEADER bfh; //位图文件头
bfh.bfReserved1 = bfh.bfReserved2 = 0;
bfh.bfType = ((WORD)('M'<< 8)'B');
bfh.bfSize = 54+size;
bfh.bfOffBits = 54;
CFile file;
if(file.Open(name,CFile::modeCreate|CFile::modeWrite))
{
 file.WriteHuge(&bfh,sizeof(BITMAPFILEHEADER)); //写入位图文件头
 file.WriteHuge(pBInfo,sizeof(BITMAPINFOHEADER)); //写入文件头
 file.WriteHuge(lpData,size); //写入数据
 file.Close(); //关闭文件
}
}
```

(10) 处理“保存”按钮的单击事件，在该事件的处理函数中获得用户设置的信息，并将信息保存到 INI 文件中，其实现代码如下：

```
void CGraspBmpDlg::OnBtsave()
{
 ::UnregisterHotKey(this->GetSafeHwnd(),HOTKEY_GRASP); //释放注册的快捷键
 WORD wvk,wmod;
 m_HotKey.GetHotKey(wvk,wmod); //获得热键控件中的键值
 BOOL result=RegisterHotKey(this->GetSafeHwnd(),HOTKEY_GRASP,wmod,wvk); //注册热键
 if(!result)
 {
 MessageBox("注册热键失败");
 return;
 }
 CString path,strwvk,strwmod,strmouse;
 m_Path.GetWindowText(path); //获得存储路径
 WritePrivateProfileString("Set","path",path,"/System.ini"); //记录存储路径
 strwvk.Format("%d",wvk); //获得虚拟键值
 WritePrivateProfileString("Set","wvk",strwvk,"/System.ini"); //记录虚拟键值
 strwmod.Format("%d",wmod); //获得修改标识
 WritePrivateProfileString("Set","wmod",strwmod,"/System.ini"); //记录修改标识
 strmouse.Format("%d",m_Cursor.GetCheck()); //获得抓取鼠标标识
 WritePrivateProfileString("Set","capmouse",strmouse,"/System.ini"); //记录鼠标标识
}
}
```

### 举一反三

根据本实例，读者可以：

- 将屏幕操作录制成 AVI；
- 远程屏幕图像发送。

## 6.4 系统相关

本节包括获得 Windows 和 System 的路径、控制光驱的弹开与关闭、启动控制面板和获取系统字体几个与系统相关的实例程序，通过这几个实例可以使读者更进一步地了解与系统相关技术。

### 实例 221

### 获得 Windows 和 System 的路径

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\06\221

### 实例说明

Windows 和 System 文件夹是系统特殊的文件夹，主要用于存放系统文件，系统中有环境变量指向这两个文件夹，应用程序在系统的任何路径下都能访问这两个文件夹下的文件，在开发应用程序时也要向这两个文件夹里复制文件。本实例实现获得这两个文件夹的完整路径。运行



程序，Windows 和 System 文件夹的完整路径就显示在文本框中，如图 6.15 所示。

### 技术要点

本实例主要应用了 GetWindowsDirectory 函数和 GetSystemDirectory 函数，下面介绍这两个函数。

GetWindowsDirectory 函数用于获得 Windows 文件夹路径名，语法如下：

```
UINT GetWindowsDirectory(LPTSTR lpBuffer,UINT uSize);
```

参数说明：

- lpBuffer：存放 Windows 文件夹路径名的缓冲区。
- uSize：缓冲区的大小。

GetSystemDirectory 函数用于获得 System 文件夹路径名，语法如下：

```
UINT GetSystemDirectory(LPTSTR lpBuffer,UINT uSize);
```

参数说明：

- lpBuffer：存放 System 文件夹路径名的缓冲区。
- uSize：缓冲区大小。

### 实现过程

- (1) 新建一个名为 GetSpeDir 的对话框 MFC 工程。
- (2) 在对话框上添加两个静态文本控件，Caption 属性分别为“system 路径”和“windows 路径”；添加两个文本编辑控件，设置 ID 属性分别为 IDC\_EDSYS DIR 和 IDC\_EDWINDIR。

(3) 主要程序代码。

```
BOOL CGetSpeDirDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 GetDlgItem(IDC_EDSYS DIR)->EnableWindow(false);
 GetDlgItem(IDC_EDWINDIR)->EnableWindow(false);
 char strwindow[50];
 char strsystem[50];
 ::GetWindowsDirectory(strwindow,50);// 获取 Windows 目录
 ::GetSystemDirectory(strsystem,50);//获取系统目录

 GetDlgItem(IDC_EDSYS DIR)->SetWindowText(strsystem);
 GetDlgItem(IDC_EDWINDIR)->SetWindowText(strwindow);
 return TRUE;
}
```

### 举一反三

根据本实例，读者可以：

- 将系统注册文件保存到系统 Windows 的路径下；
- 将加密文件的密钥信息保存到 System 的路径下。

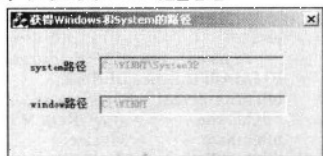


图 6.15 获得 Windows 和 System 的路径

## 实例 222 控制光驱的弹开与关闭

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\06\222

### 实例说明

有些媒体播放器程序提供了弹出和关闭光驱的功能（如超级解霸）。本实例实现控制光驱打开和关闭。运行程序，如图 6.16 所示，单击“打开光驱”按钮，可使光驱弹出；单击“关闭光驱”可关闭光驱。

## 技术要点

本实例主要使用 `mciSendString` 函数实现光驱打开和关闭,该函数是 `winmm.lib` 库中的函数, `winmm.lib` 库中包含着与多媒体有关的函数, `mciSendString` 函数用于向多媒体设备发送命令语句,语法如下:

```
MCIERROR mciSendCommand(MCIDEVICEID IDDevice,
 UINT uMsg,DWORD fdwCommand,DWORD dwParam);
```

参数说明:

- `IDDevice`: 已经打开的多媒体设备。
- `uMsg`: 命令消息。
- `fdwCommand`: 命令消息的标识。
- `dwParam`: 命令消息的参数。

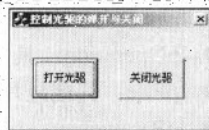


图 6.16 控制光驱的弹出与关闭

## 实现过程

- (1) 新建一个名为 `ControlCdrum` 的对话框 MFC 工程。
- (2) 在对话框上添加两个按钮控件,设置 ID 属性分别是 `IDC_BTOPEN` 和 `IDC_BTCLSE`,设置 Caption 属性分别是“打开光驱”和“关闭光驱”。

(3) “打开光驱”按钮的实现代码:

```
void CControlCdrumDlg::OnOpen()
{
 mciSendString("set cdaudio door open",0,0,NULL);
}
```

(4) “关闭光驱”按钮的实现代码:

```
void CControlCdrumDlg::OnClose()
{
 mciSendString("set cdaudio door closed",0,0,NULL);
}
```

## 举一反三

根据本实例,读者可以:

- 在软件安装完毕时自动弹出光驱。

## 实例 223 启动控制面板

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\06\223

## 实例说明

在 Windows 系统中,经常要打开控制面板以进行相应的设置。本实例实现在程序中调用控制面板中相关信息设置的对话框。运行程序,如图 6.17 所示,单击程序中的不同按钮,其实现效果就如同使用控制面板一样。

## 技术要点

本实例主要通过 `ShellExecute` 函数实现调用控制面板设置对话框, `ShellExecute` 函数和 `Winexec` 函数一样都是调用其他应用程序来执行,其语法如下:

```
HINSTANCE ShellExecute(HWND hwnd, LPCTSTR lpOperation,
 LPCTSTR lpFile, LPCTSTR lpParameters, LPCTSTR lpDirectory,INT nShowCmd);
```

参数说明:



图 6.17 启动控制面板

- hwnd: 应用程序窗体句柄。
- lpOperation: 具体操作命令, 如 open, print, explore。
- lpFile: 其他应用程序文件名。
- lpParameters: 应用程序调用的参数。
- lpDirectory: 默认的目录地址。
- nShowCmd: 窗体显示参数, 如 SW\_SHOW。

调用控制面板相关的设置对话框主要是执行 rundll32.exe 程序。例如打开 IE 浏览器设置窗口, 可以选择“开始/运行”菜单命令, 在“打开”文本框中输入 rundll32.exe shell32.dll Control\_RunDLL inetctl.cpl 语句。

## 实现过程

- (1) 新建一个名为 ControlPanl 的对话框 MFC 工程。
- (2) 在对话框上添加 12 个按钮控件。
- (3) 主要程序代码。

```
void CControlPanlDlg::OnButton1()
{
 //打开 IE4 浏览器的设置窗口
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL inetctl.cpl",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton2()
{
 //打开声音的设置窗口
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL mmsys.cpl @1",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton3()
{
 //启动日期和时间设置
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL timedate.cpl",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton4()
{
 //启动显示设置面板
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL desk.cpl",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton5()
{
 //启动辅助选项
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL access.cpl",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton6()
{
 //打开鼠标设置
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL main.cpl @0",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton7()
{
 //启动键盘设置
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL main.cpl @1",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton8()
{
 //打开区域设置
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
```

```

"shell32.dll Control_RunDLL intl.cpl",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton9()
{
 //启动添加软件设置
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL appwiz.cpl",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton10()
{
 //启动添加硬件设置
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL hdwwiz.cpl",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton11()
{
 //打开系统设置
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL sysdm.cpl",NULL,SW_SHOW);
}

void CControlPanlDlg::OnButton12()
{
 //启动调制解调器设置
 ::ShellExecute(NULL,"OPEN","rundll32.exe",
 "shell32.dll Control_RunDLL modem.cpl",NULL,SW_SHOW);
}

```

### 举一反三

根据本实例，读者可以：

- 在应用程序中调用“日期/时间”调整系统时间；
- 在应用软件中调用“添加/删除程序”删除程序。

## 实例 224 定时关闭计算机

本实例可以提高基础技能

实例位置：光盘\mingrisoft\06\224

### 实例说明

很多人在使用手机时都会设置时间定时关机，这样可以确保晚上休息时不被打扰，那么这种功能对于电脑是否适用呢，当然是可以的，本例就是使用 Visual C++ 来实现定时关闭计算机的功能。运行本实例，首先设置关机时间，选择是否开机运行本程序，单击“设置”按钮，即可将用户的设置保存到 INI 文件中，然后程序会根据用户设置的关机时间关闭计算机。实例运行结果如图 6.18 所示。

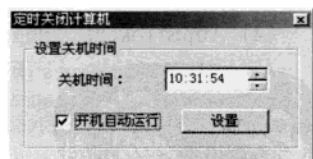


图 6.18 定时关闭计算机

要实现定时关闭计算机，首先要设置一个定时器，设置间隔为 1s，在定时器的处理函数中通过 CTime 类获得系统时间，然后比较用户设置的时间和系统当前时间是否相同，如果相同则调用 ExitWindowsEx 函数关闭计算机，为了使程序不影响用户的其他工作，可以将程序最小化到系统托盘中，并且为了方便用户的使用，可以设置开机自动运行程序，也就是说，当用户选择开机自动运行以后，只要设置一次关机时间，就可以每天都有效了。

### 技术要点

本示例中设计定时关闭计算机程序，主要用到了 ExitWindowsEx 函数、OpenProcessToken



函数、LookupPrivilegeValue 函数和 AdjustTokenPrivileges 函数以及 CRegKey 类的方法，下面对本实例中用到的关键技术进行详细讲解。

#### (1) ExitWindowsEx 函数

ExitWindowsEx 函数用于关闭或者重启计算机。其语法格式如下：

```
BOOL ExitWindowsEx(UINT uFlags, DWORD dwReserved);
```

参数说明：

- uFlags：指定进行的操作，可选值如表 6.1 所示。

表 6.1 uFlags 参数值表

| 参 数 值        | 描 述       |
|--------------|-----------|
| EWX_LOGOFF   | 中止进程，然后注销 |
| EWX_POWEROFF | 关闭系统      |
| EWX_REBOOT   | 重新启动系统    |
| EWX_SHUTDOWN | 关掉系统电源    |

- dwReserved：保留变量，必须为 0。

#### (2) OpenProcessToken 函数

OpenProcessToken 函数用于获得应用程序权限的令牌，其语法格式如下：

```
BOOL OpenProcessToken(HANDLE ProcessHandle, DWORD DesiredAccess, PHANDLE TokenHandle);
```

参数说明：

- ProcessHandle：获得令牌的应用程序句柄。
- DesiredAccess：访问令牌的请求类型。
- TokenHandle：令牌句柄。

#### (3) LookupPrivilegeValue 函数

LookupPrivilegeValue 函数用于获得权限的惟一标识值，其语法格式如下：

```
BOOL LookupPrivilegeValue(LPCTSTR lpSystemName, LPCTSTR lpName, PLUID lpLuid);
```

参数说明：

- lpSystemName：权限所在的系统名称。
- lpName：权限的名称。
- lpLuid：权限的惟一标识。

#### (4) AdjustTokenPrivileges 函数

AdjustTokenPrivileges 函数用于设置令牌的权限，其语法格式如下：

```
BOOL AdjustTokenPrivileges(HANDLE TokenHandle, BOOL DisableAllPrivileges, PTOKEN_PRIVILEGES NewState, DWORD BufferLength, PTOKEN_PRIVILEGES PreviousState, PDWORD ReturnLength);
```

参数说明：

- TokenHandle：令牌句柄。
- DisableAllPrivileges：打开或关闭令牌权限。
- NewState：新的 TOKEN\_PRIVILEGES 结构对象，TOKEN\_PRIVILEGES 结构包含着一些权限信息及权限的属性。
- BufferLength：上一个 TOKEN\_PRIVILEGES 结构对象的缓存大小。
- PreviousState：上一个 TOKEN\_PRIVILEGES 结构对象。
- ReturnLength：返回上一个 TOKEN\_PRIVILEGES 结构对象所要求的缓存大小。

#### (5) CRegKey 类的方法

CRegKey 是 MFC 类库中操作注册表的类，该类的主要方法及变量如表 6.2 所示。

表 6.2 CRegKey 类的主要方法及变量表

| 方法及成员变量          | 描 述                                         |
|------------------|---------------------------------------------|
| Attach           | 返回 HKEY 句柄的 CRegKey 对象, 设置 m_hKey 为 HKEY 对象 |
| Close            | 关闭已打开的 CRegKey 对象                           |
| Create           | 新建 CRegKey 对象                               |
| CRegKey          | CRegKey 类的构造函数, 设置 m_hKey 为 NULL            |
| DeleteSubKey     | 删除注册表项                                      |
| DeleteValue      | 删除串值                                        |
| Detach           | 使 CRegKey 对象从 HKEY 句柄中分离, 设置 m_hKey 为 NULL  |
| Open             | 打开注册表项                                      |
| QueryValue       | 查找注册表中串的值                                   |
| RecurseDeleteKey | 删除注册表项及子项                                   |
| SetKeyValue      | 设置注册表项的值                                    |
| SetValue         | 设置注册表中串的值                                   |
| operator HKEY    | 从 CRegKey 对象中得到 HKEY 句柄                     |
| m_hKey           | CRegKey 类的成员变量                              |

## 实现过程

(1) 新建一个基于对话框的应用程序, 将其窗体标题改为“定时关闭计算机”。

(2) 向工程中导入一个图标资源, 并向对话框中添加一个群组控件、一个静态文本控件、一个时间控件、一个复选框控件和一个按钮控件。对话框中主要用到的控件及说明如表 6.3 所示。

表 6.3 对话框主要用到的控件及说明

| 控件 ID               | 属 性 设 置         | 关 联 变 量             |
|---------------------|-----------------|---------------------|
| IDC_DATETIMEPICKER1 | Format: Time    | COleDateTime m_Time |
| IDC_CHECK1          | Caption: 开机自动运行 | CButton m_Check     |
| IDC_BUTTONSET       | Caption: 设置     | 无                   |

(3) 主要程序代码。

在主窗体的头文件中, 声明变量, 代码如下:

```
CString m_Skey; //注册表键值
HICON m_Icon; //托盘图标
```

在主窗体初始化时设置用户权限, 并读取用户在 INI 文件中的设置, 其实现代码如下:

```
m_Icon = AfxGetApp()->LoadIcon(IDI_ICON1); //加载图标资源
HANDLE hToken;
OpenProcessToken(GetCurrentProcess(),TOKEN_ADJUST_PRIVILEGES|TOKEN_QUERY,&hToken); //打开进程标识
TOKEN_PRIVILEGES Privilege;
LookupPrivilegeValue(NULL,"Seshutdownprivilege",&Privilege.Privileges[0].Luid); //查找权限
Privilege.PrivilegeCount = 1;
Privilege.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
DWORD dwLen;
AdjustTokenPrivileges(hToken,false,&Privilege,sizeof(TOKEN_PRIVILEGES),NULL,&dwLen); //更改权限
m_Skey="Software\\Microsoft\\Windows\\CurrentVersion\\Run";
ModifyStyleEx(WS_EX_APPWINDOW,WS_EX_TOOLWINDOW);
//添加系统托盘
char lpszTip[256];
strcpy(lpszTip,"屏幕录像"); //托盘提示字符
```

```

NOTIFYICONDATA data;
data.cbSize = sizeof(NOTIFYICONDATA); //结构的大小
data.hWnd = m_hWnd; //接收消息窗口句柄
lstrcpy(data.szTip,lpszTip,sizeof(lpszTip)); //提示信息
data.uCallbackMessage = WM_ONTRAY; //消息响应函数
data.uFlags = NIF_MESSAGE | NIF_ICON | NIF_TIP; //设置结构成员有效
data.uID = IDR_MAINFRAME; //图标标识
data.hIcon = m_hIcon; //图标句柄
Shell_NotifyIcon(NIM_ADD,&data); //添加系统托盘
CString str;
char buf[2];
GetPrivateProfileString("设置","开机运行","",buf,2,"./Setting.ini"); //读取开机运行标志
str = buf;
m_Check.SetCheck(atoi(str)); //设置复选框状态
GetPrivateProfileString("设置","是否关机","",buf,2,"./Setting.ini"); //读取是否关机标志
str = buf;
if(str == "1") //如果设置了关机
{
 SetTimer(1,1000,NULL); //开启定时器
 char time[8];
 GetPrivateProfileString("设置","时间设置","",time,8,"./Setting.ini"); //获得关机时间
 str = time;
 m_Time.SetTime(atoi(str.Left(2)),atoi(str.Mid(2,2)),atoi(str.Right(2))); //根据关机时间设置时间控件
 UpdateData(FALSE); //更新控件
}

```

处理“设置”按钮的单击事件，在该事件的处理函数中将用户的设置信息保存到 INI 文件和注册表中，其实现代码如下：

```

void CCloseTimerDlg::OnButtonset()
{
 UpdateData(); //更新数据交换
 if(m_Check.GetCheck()) //如果设置开机运行
 {
 WritePrivateProfileString("设置","开机运行","1","./Setting.ini"); //写入到INI文件中
 char path[256];
 ::GetCurrentDirectory(256,path); //获得程序当前目录
 CString strname;
 strname.Format("%s\\Debug\\CloseTimer.exe",path); //设置可执行文件路径
 CRegKey writevalue;
 writevalue.Create(HKEY_CURRENT_USER,m_Skey); //打开注册表中的键
 writevalue.SetValue(strname,"TimerRun"); //设置开机运行
 writevalue.Close(); //关闭CRegKey对象
 }
 else //如果不是开机运行
 {
 WritePrivateProfileString("设置","开机运行","0","./Setting.ini"); //写入到INI文件中
 CRegKey delkey;
 if(!delkey.Open(HKEY_CURRENT_USER,m_Skey)) //打开注册表项
 {
 delkey.DeleteValue("TimerRun"); //删除自动运行的键值
 delkey.Close(); //关闭CRegKey对象
 }
 WritePrivateProfileString("设置","是否关机","1","./Setting.ini"); //在INI文件中写入关机标志
 WritePrivateProfileString("设置","时间设置",m_Time.Format("%H%M%S"),"./Setting.ini"); //写入关机时间
 KillTimer(1); //关闭定时器
 SetTimer(1,1000,NULL); //重新开启定时器
 }
}

```

手动添加 WM\_ONTRAY 消息的映射和处理函数，在该消息的处理函数中设置用户在托盘图标中的双击和鼠标右键单击功能，其实现代码如下：

```

void CCloseTimerDlg::OnTray(WPARAM wParam, LPARAM lParam)
{
 if(lParam == WM_LBUTTONDOWNBLCLK) //双击鼠标
 {
 ShowWindow(SW_RESTORE); //恢复窗口的显示
 }
}

```

```

if(lpParam == WM_RBUTTONDOWN) //双击鼠标
{
 //弹出消息框, 根据用户选择判断是否退出程序
 if(MessageBox("确定要退出程序吗?", "系统提示", MB_OKCANCEL|MB_ICONQUESTION) == IDOK)
 {
 //删除系统托盘
 NOTIFYICONDATA data;
 data.cbSize = sizeof(NOTIFYICONDATA);
 data.hWnd = m_hWnd;
 data.hIcon = m_hIcon;
 Shell_NotifyIcon(NIM_DELETE, &data);
 CDialog::OnCancel();
 }
}

```

处理主窗体的 WM\_CLOSE 事件。在该事件的处理函数中设置主窗体隐藏, 其实现代码如下:

```

void CCloseTimerDlg::OnClose()
{
 ShowWindow(SW_HIDE); //隐藏主窗体
 //CDialog::OnClose(); //不退出程序
}

```

### 举一反三

根据本实例, 读者可以:

- 重启计算机。

### 实例 225

### 实现 OCX 控件的注册和卸载

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\06\225

### 实例说明

由于 OCX 控件能够在各种编程语言中使用, 因此使得 OCX 控件被广泛地应用于程序中。在程序中使用 OCX 控件需要进行注册, 否则是不允许使用 OCX 控件的。在各种编程的集成开发环境中都提供了注册和卸载 OCX 控件的功能, 本例就是使用 Visual C++ 来实现 OCX 控件的注册和卸载。运行本实例, 单击“...”按钮, 在弹出的“打开”对话框中选择一个 OCX 控件, 然后单击“注册”按钮和“卸载”按钮可以注册和卸载当前选择的 OCX 控件。程序运行如图 6.19 所示。

### 技术要点

本示例中实现注册和卸载 OCX 控件的功能时, 主要用到了 LoadLibrary 函数、FreeLibrary 函数、DllRegister Server 函数和 DllUnregisterServer, 下面对本实例中用到的关键技术进行详细讲解。

(1) LoadLibrary 函数。LoadLibrary 函数用于载入指定的动态链接库, 并将它映射到当前进程使用的地址空间。一旦载入, 即可访问库内保存的资源。其语法格式如下:

HINSTANCE LoadLibrary(LPCTSTR lpLibFileName);

lpLibFileName: 指定要载入的动态链接库的名称。

(2) FreeLibrary 函数。FreeLibrary 函数用于释放大 LoadLibrary 函数装载的动态链接库, 其语法格式如下:

BOOL FreeLibrary(HMODULE hLibModule);

hLibModule: 要释放的动态链接库句柄。

(3) DllRegisterServer 函数。DllRegisterServer 函数用于注册 OCX 控件, 其语法格式如下:

STDAPI DllRegisterServer(void);

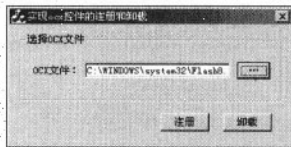


图 6.19 实现 OCX 控件的注册和卸载



(4) DllUnregisterServer 函数。DllUnregisterServer 函数用于卸载 OCX 控件，其语法格式如下：

```
STDAPI DllUnregisterServer(void);
```

## 实现过程

- (1) 新建一个基于对话框的应用程序，将其窗体标题改为“实现 OCX 控件的注册和卸载”。
- (2) 向对话框中添加 1 个群组控件、1 个静态文本控件、1 个编辑框控件和 3 个按钮控件。
- (3) 处理“...”按钮的单击事件，在该事件的处理函数中调用“打开”对话框选择一个 OCX 控件，其实现代码如下：

```
void CRegisterOcxDlg::OnButpath()
{
 //构造打开对话框
 CFileDialog fDlg(TRUE,NULL,NULL,OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,"OCX文件*.ocx",this);
 if(fDlg.DoModal() == IDOK) //显示打开对话框
 {
 m_Path.SetWindowText(fDlg.GetPathName()); //获得OCX控件路径
 }
}
```

- (4) 处理“注册”按钮的单击事件，在该事件的处理函数中注册用户所选择的 OCX 控件，其实现代码如下：

```
void CRegisterOcxDlg::OnButregocx()
{
 CString fName;
 m_Path.GetWindowText(fName); //获得OCX控件路径
 if(!fName.IsEmpty()) //如果路径不为空
 {
 HMODULE hModule = LoadLibrary(fName); //加载动态链接库
 if(hModule == NULL)
 {
 return;
 }
 //获得DllRegisterServer地址
 pFunDllRegisterServer DllRegisterServer = (pFunDllRegisterServer)GetProcAddress(hModule, "DllRegister Server");
 if(DllRegisterServer != NULL)
 {
 HRESULT ret = DllRegisterServer(); //注册OCX控件
 if(ret == S_OK) //如果成功
 MessageBox("注册成功"); //提示注册成功
 else //否则
 MessageBox("注册失败"); //提示注册失败
 }
 FreeLibrary(hModule); //释放动态链接库
 }
}
```

- (5) 处理“卸载”按钮的单击事件，在该事件的处理函数中卸载用户所选择的 OCX 控件，其实现代码如下：

```
void CRegisterOcxDlg::OnButunregocx()
{
 CString fName;
 m_Path.GetWindowText(fName); //获得OCX控件路径
 if(!fName.IsEmpty()) //如果路径不为空
 {
 HMODULE hModule = LoadLibrary(fName); //加载动态链接库
 if(hModule == NULL)
 {
 return;
 }
 pFunDllRegisterServer DllUnRegisterServer = (pFunDllRegisterServer)GetProcAddress(hModule, "DllUnregisterServer"); //获得DllUnregisterServer地址
 if(DllUnRegisterServer != NULL)
 {
 HRESULT ret = DllUnRegisterServer(); //卸载OCX控件
 if(ret == S_OK) //如果卸载成功
 MessageBox("卸载成功"); //提示卸载成功
 else //否则
 MessageBox("卸载失败"); //提示卸载失败
 }
 FreeLibrary(hModule);
 }
}
```

## 举一反三

根据本实例，读者可以：

- 将 OCX 控件注册与卸载功能添加到右键菜单。

## 6.5 系统监控

本节主要通过检测系统启动模式、内存使用状态和监视剪贴板内容等几个实例介绍系统监控的相关内容，通过系统监控可以实现应用程序开发过程中的辅助功能。

## 实例 226 检测 U 盘是否插入

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\06\226

## 实例说明

检测 U 盘是否插入可使用 WM\_DEVICECHANGE 消息来获取，通过截获这个消息同时获取驱动器列表就可以看到插入和拔除的驱动器设备。程序运行如图 6.20 所示。

## 技术要点

WM\_DEVICECHANGE 消息截获时需要两个结构变量 DEV\_BROADCAST\_HDR 和 DEV\_BROADCAST\_VOLUME，第一个结构代表了驱动设备的信息头，第二个结构代表了驱动设备的值。这两个结构的定义如下：

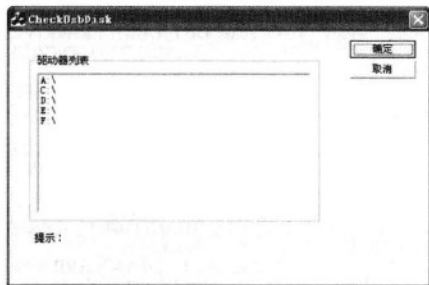


图 6.20 检测 U 盘是否插入

```
typedef struct _DEV_BROADCAST_HDR {
 ULONG dbch_size; //结构大小
 ULONG dbch_devicetype; //设备类型
 ULONG dbch_reserved;
} DEV_BROADCAST_HDR;
typedef DEV_BROADCAST_HDR *PDEV_BROADCAST_HDR;

typedef struct _DEV_BROADCAST_VOLUME {
 ULONG dbcv_size; //结构大小
 ULONG dbcv_devicetype; //设备类型
 ULONG dbcv_reserved;
 ULONG dbcv_unitmask; //设备索引
 USHORT dbcv_flags; //本地还是网络
} DEV_BROADCAST_VOLUME;
typedef DEV_BROADCAST_VOLUME *PDEV_BROADCAST_VOLUME;
```

## 实现过程

- (1) 新建一个名为 CheckUsbDisk 的对话框 MFC 工程。
- (2) 在对话框上添加一个静态文本控件，设置 Caption 属性为“提示：”；添加一个列表框控件；添加两个按钮控件，设置 Caption 属性为“确定”和“取消”。
- (3) 定义获取驱动器列表的方法，并将驱动设备信息显示在列表框中：

```
void CCheckUsbDiskDlg::UpdateDeviceList()
{//显示驱动器列表
 m_listbox.ResetContent();
 size_t szAllDriveStrings = GetLogicalDriveStrings(0, NULL);
 char *pDriveStrings = new char[szAllDriveStrings + sizeof(T(" "));
 GetLogicalDriveStrings(szAllDriveStrings, pDriveStrings);
 size_t szDriveString = strlen(pDriveStrings);
 while(szDriveString > 0)
```



```

 {
 m_listbox.AddString(pDriveStrings);
 pDriveStrings += szDriveString + 1;
 szDriveString = strlen(pDriveStrings);
 }
}

```

(4) 在 DefWindowProc 消息事件中截获驱动器发生改变时的消息。实现代码如下:

```

LRESULT CCheckUsbDiskDlg::DefWindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
 if(message == WM_DEVICECHANGE) //0x8000,0x8004
 {
 CString str;
 DEV_BROADCAST_HDR* dhr = (DEV_BROADCAST_HDR*)lParam;
 switch(wParam)
 {
 case DBT_CONFIGCHANGECANCELED:
 TRACE("DBT_CONFIGCHANGECANCELED");
 break;
 case DBT_CONFIGCHANGED:
 TRACE("DBT_CONFIGCHANGED");
 break;
 case DBT_DEVICEQUERYREMOVE:
 TRACE("DBT_DEVICEQUERYREMOVE");
 break;
 case DBT_DEVICEQUERYREMOVEFAILED:
 TRACE("DBT_DEVICEQUERYREMOVEFAILED");
 break;
 case DBT_DEVICEREMOVEPENDING:
 TRACE("DBT_DEVICEREMOVEPENDING");
 break;
 case DBT_DEVICETYPESPECIFIC:
 TRACE("DBT_DEVICETYPESPECIFIC");
 break;
 case DBT_QUERYCHANGECONFIG:
 TRACE("DBT_QUERYCHANGECONFIG");
 break;
 case DBT_USERDEFINED:
 TRACE("DBT_USERDEFINED");
 break;
 case DBT_DEVICEARRIVAL://插入设备
 if(dhr->dbch_devicetype == DBT_DEVTYP_VOLUME)
 {
 PDEV_BROADCAST_VOLUME lpdbv = (PDEV_BROADCAST_VOLUME)dhr;
 if(lpdbv->dbcv_flags & DBTF_MEDIA)
 {
 UpdateDeviceList();
 str.Format("提示: 设备 %c 插入",
 FirstDriveFromMask(lpdbv->dbcv_unitmask)); //AfxMessageBox(str);
 m_text.SetWindowText(str);
 }
 //else
 //{
 char ch = FirstDriveFromMask(lpdbv->dbcv_unitmask);
 str.Format("%c: \\",ch);
 //}
 }
 break;
 case DBT_DEVICEREMOVECOMPLETE://拔出设备
 if(dhr->dbch_devicetype == DBT_DEVTYP_VOLUME)
 {
 PDEV_BROADCAST_VOLUME lpdbv = (PDEV_BROADCAST_VOLUME)dhr;
 if(lpdbv->dbcv_flags & DBTF_MEDIA)
 {
 UpdateDeviceList();
 str.Format("提示: 设备 %c 拔除",
 FirstDriveFromMask(lpdbv->dbcv_unitmask));
 m_text.SetWindowText(str);
 }
 //else
 //{
 str.Format("Drive %c 拔除",
 FirstDriveFromMask(lpdbv->dbcv_unitmask));
 //}
 }
 break;
 default:
 break;
 }
 }
}

```

```
 }
 return CDialog::DefWindowProc(message, wParam, lParam);
}
```

(5) 实现 FirstDriveFromMask 函数，该函数用来根据驱动器索引返回驱动器名称。实现代码如下：

```
char FirstDriveFromMask (ULONG unitmask)
{
 char i;

 for (i = 0; i < 26; ++i)
 {
 if (unitmask & 0x1)
 break;
 unitmask = unitmask >> 1;
 }

 return (i + 'A');
}
```

### 举一反三

根据本实例，读者可以：

- 检测所有驱动设备。

## 实例 227 检测文件和目录是否改变

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\06\227

### 实例说明

在许多时候我们需要检测指定目录下的文件是否发生了如新建、修改和删除等变化。本实例实现了实时检测指定文件和目录是否发生了改变的功能，并将检测结果显示在列表控件中，如图 6.21 所示。

### 技术要点

获取文件或目录的改变信息通过 API 函数 ReadDirectoryChangesW 实现，此函数的定义如下：

```
BOOL ReadDirectoryChangesW(
 HANDLE hDirectory,
 LPVOID lpBuffer,
 DWORD nBufferLength,
 BOOL bWatchSubtree,
 DWORD dwNotifyFilter,
 LPDWORD lpBytesReturned,
 LPOVERLAPPED lpOverlapped,
 LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

参数说明如下：

- hDirectory：文件或目录的句柄。
- lpBuffer：获取数据缓冲区的指针。
- nBufferLength：获取数据缓冲区的长度。
- bWatchSubtree：是否搜索子目录。
- dwNotifyFilter：检测更改的类型。
- lpBytesReturned：返回所检测数据的大小。
- lpOverlapped：此参数异步操作时使用，可设为空。
- lpCompletionRoutine：此参数在检测完成时提供一个回调函数，可设为空。



图 6.21 检测文件和目录是否改变



## 实现过程

(1) 新建一个名为 DirChange 的对话框 MFC 工程。

(2) 在对话框上添加 1 个静态文本控件, 设置标题为“选择目录”; 添加 1 个文本编辑框控件; 添加 3 个按钮控件, 分别设置标题为“选择目录”、“开始”和“结束”; 添加 1 个列表视图控件用来显示检测结果。

(3) 选择检测的目录, 代码如下:

```
bool CDirChangeDlg::OpenDir(char *pStartPath)
{
 LPMAALLOC pMalloc;
 if(SHGetMalloc(&pMalloc)!=NOERROR)
 return false;
 /*if(pStartPath && IsValuePath(pStartPath))
 {
 char *pStr=m_pDirPath;
 while(*pStr++!=*pStartPath++);
 }*/
 BROWSEINFO bInfo={this->m_hWnd,NULL,0,"请选择文件夹:",0,NULL,NULL,0};
 ITEMIDLIST *pItemLst=SHBrowseForFolder(&bInfo);
 if(pItemLst)
 {
 SHGetPathFromIDList(pItemLst,pStartPath);
 pMalloc->Free(pItemLst);
 pMalloc->Release();
 return TRUE;
 }
 pMalloc->Free(pItemLst);
 pMalloc->Release();
 return false;
}
```

(4) 目录及文件的检测是在单独的线程中完成的, 这里只给出线程函数的实现代码, 代码如下:

```
DWORD WINAPI CDirChangeDlg::ThreadProc(LPVOID lParam)
{
 CDirChangeDlg* obj = (CDirChangeDlg*)lParam; //参数转化

 obj->hDir = CreateFile(//打开目录, 得到目录的句柄
 obj->m_strWatchedDir,
 GENERIC_READ|GENERIC_WRITE,
 FILE_SHARE_READ|FILE_SHARE_WRITE|FILE_SHARE_DELETE,
 NULL,
 OPEN_EXISTING,
 FILE_FLAG_BACKUP_SEMANTICS,
 NULL
);
 if(obj->hDir ==INVALID_HANDLE_VALUE)
 return false;

 char buf[(sizeof(FILE_NOTIFY_INFORMATION)+MAX_PATH)*2];
 FILE_NOTIFY_INFORMATION* pNotify=(FILE_NOTIFY_INFORMATION*)buf;
 DWORD dwBytesReturned;
 while(true)
 { //指定获取目录变更信息
 if(::ReadDirectoryChangesW(obj->hDir,
 pNotify,
 sizeof(buf),
 true,
 FILE_NOTIFY_CHANGE_FILE_NAME|
 FILE_NOTIFY_CHANGE_DIR_NAME|
 FILE_NOTIFY_CHANGE_ATTRIBUTES|
 FILE_NOTIFY_CHANGE_SIZE|
 FILE_NOTIFY_CHANGE_LAST_WRITE|
 FILE_NOTIFY_CHANGE_LAST_ACCESS|
 FILE_NOTIFY_CHANGE_CREATION|
 FILE_NOTIFY_CHANGE_SECURITY,
 &dwBytesReturned,
 NULL,
 NULL))
 {
 char tmp[MAX_PATH], str1[MAX_PATH], str2[MAX_PATH];
 memset(tmp, 0, sizeof(tmp));
 WideCharToMultiByte(CP_ACP,0,pNotify->FileName,pNotify->FileNameLength/2,tmp,99,NULL,NULL);
 strepy(str1, tmp);
```

```

if(pNotify->NextEntryOffset != 0)
{
 PFILE_NOTIFY_INFORMATION p =
(PFILE_NOTIFY_INFORMATION)((char*)pNotify->NextEntryOffset);
 memset(tmp, 0, sizeof(tmp));
 WideCharToMultiByte(CP_ACP,0,p->FileName,p->FileNameLength/2,tmp,99,NULL,NULL);
 strcpy(str2, tmp);
}
switch(pNotify->Action)
{
 case FILE_ACTION_ADDED://添加新文件
 {
 CTime tt=CTime::GetCurrentTime();
 CString strTT;
 strTT.Format("%d:%d:%d",tt.GetHour(),tt.GetMinute(),tt.GetSecond());
 obj->m_list.InsertItem(0,obj->m_szi);
 obj->m_list.SetItemText(0,2,"添加了新文件");
 obj->m_list.SetItemText(0,3,str1);
 obj->m_list.SetItemText(0,1,strTT);
 }
 break;

 case FILE_ACTION_REMOVED://文件删除
 {
 CTime tt=CTime::GetCurrentTime();
 CString strTT;
 strTT.Format("%d:%d:%d",tt.GetHour(),tt.GetMinute(),tt.GetSecond());
 obj->m_list.InsertItem(0,obj->m_szi);
 obj->m_list.SetItemText(0,2,"删除了文件");
 obj->m_list.SetItemText(0,3,str1);
 obj->m_list.SetItemText(0,1,strTT);
 }
 break;

 case FILE_ACTION_RENAMED_NEW_NAME://文件重命名
 {
 CTime tt=CTime::GetCurrentTime();
 CString strTT;
 strTT.Format("%d:%d:%d",tt.GetHour(),tt.GetMinute(),tt.GetSecond());
 obj->m_list.InsertItem(0,obj->m_szi);
 obj->m_list.SetItemText(0,2,"重命名了文件");
 strcat(str1,"->");
 obj->m_list.SetItemText(0,3,strcat(str1,str2));
 obj->m_list.SetItemText(0,1,strTT);
 }
 break;

 case FILE_ACTION_RENAMED_OLD_NAME://文件重命名
 {
 CTime tt=CTime::GetCurrentTime();
 CString strTT;
 strTT.Format("%d:%d:%d",tt.GetHour(),tt.GetMinute(),tt.GetSecond());
 obj->m_list.InsertItem(0,obj->m_szi);
 obj->m_list.SetItemText(0,2,"重命名了文件");
 strcat(str1," 改名为 ");
 obj->m_list.SetItemText(0,3,strcat(str1,str2));
 obj->m_list.SetItemText(0,1,strTT);
 }
 break;

 case FILE_ACTION_MODIFIED://文件修改
 {
 CTime tt=CTime::GetCurrentTime();
 CString strTT;
 strTT.Format("%d:%d:%d",tt.GetHour(),tt.GetMinute(),tt.GetSecond());
 obj->m_list.InsertItem(0,obj->m_szi);
 obj->m_list.SetItemText(0,2,"修改了文件");
 obj->m_list.SetItemText(0,3,str1);
 obj->m_list.SetItemText(0,1,strTT);
 }
 break;

 default:
 {
 CTime tt=CTime::GetCurrentTime();
 CString strTT;
 strTT.Format("%d:%d:%d",tt.GetHour(),tt.GetMinute(),tt.GetSecond());
 obj->m_list.InsertItem(0,obj->m_szi);
 obj->m_list.SetItemText(0,2,"未知变化");
 obj->m_list.SetItemText(0,3,"");
 }
}

```



```

 obj->m_list.SetItemText(0,1,strTT);
 }
 break;
}
obj->m_i++;
itoa(obj->m_i,obj->m_szi,10);
}
else
 break;
}
return 0;
}

```

### 举一反三

根据本实例，读者可以：

- 制作文件式电报接收程序。

## 实例 228 检测系统启动模式

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\06\228

### 实例说明

在启动计算机时按下〈F8〉键会出现多种启动计算机的方式，包括正常启动、安全模式、带网络的安全模式等。本例可以确定计算机当前的启动模式。运行程序，系统的启动模式显示在文本框中，如图 6.22 所示。

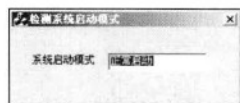


图 6.22 检测系统启动模式

### 技术要点

本实例使用 GetSystemMetrics 函数获得与 Windows 环境有关的信息，其语法如下：

```
int GetSystemMetrics(int nIndex);
```

参数说明：

- nIndex：指定想要获取信息的标识。

### 实现过程

- (1) 新建一个名为 StartMode 的对话框 MFC 工程。
- (2) 在对话框上添加一个静态文本控件，设置 Caption 属性为“系统启动模式”；添加文本编辑控件，设置 ID 属性为 IDC\_EDSTARTMODE。
- (3) 主要程序代码。

```

BOOL CStartModeDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 int i=::GetSystemMetrics(SM_CLEANBOOT);
 switch(i)
 {
 case 0:
 GetDlgItem(IDC_EDSTARTMODE)->SetWindowText("正常启动");
 break;
 case 1:
 GetDlgItem(IDC_EDSTARTMODE)->SetWindowText("安全模式");
 break;
 }

 return TRUE;
}

```

### 举一反三

根据本实例，读者可以：

- 根据启动模式控制程序的执行界面；

● 系统启动模式提示程序。

## 实例 229 内存使用状态

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\06\229

### 实例说明

内存是计算机的主要部件，内存不足可能会使计算机死机，如果对计算机的内存使用状态进行跟踪，将能避免死机现象的发生。本实例实现了获得当前内存使用情况。运行程序，内存的相关信息就显示在文本框中，如图 6.23 所示。

### 技术要点

本实例应用 GlobalMemoryStatus 函数来获得内存使用的情况，其语法如下：

```
VOID GlobalMemoryStatus(LPMEMORYSTATUS lpBuffer);
```

参数说明：

- lpBuffer：MEMORYSTATUS 结构对象指针，MEMORYSTATUS 结构包含当前状态的物理内存和虚拟内存信息，结构成员如下：
- dwLength：结构的长度。
- dwMemoryLoad：内存使用的百分比。
- dwTotalPhys：物理内存总量。
- dwAvailPhys：可用的物理内存。
- dwTotalPageFile：内存页总量。
- dwAvailPageFile：可用内存页数量。
- dwTotalVirtual：虚拟内存空间总量。
- dwAvailVirtual：可用虚拟内存。

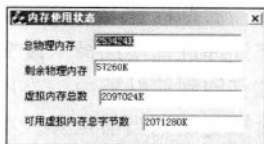


图 6.23 内存使用状态

### 实现过程

- (1) 新建一个名为 MomoryState 的对话框 MFC 工程。
- (2) 在对话框上添加 4 个静态文本控件；添加 4 个文本编辑控件，设置 ID 属性分别为 IDC\_EDPHYTOTAL、IDC\_EDPHYFREE、IDC\_EDVIRTOTAL 和 IDC\_EDVIRFREE。

(3) 主要程序代码。

```
BOOL CMemoryStateDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略

 MEMORYSTATUS memory;
 CString strmem;
 ::GlobalMemoryStatus(&memory); //获取内存信息
 strmem.Format("%dK", memory.dwTotalPhys/1024);
 GetDlgItem(IDC_EDPHYTOTAL)->SetWindowText(strmem);
 strmem.Format("%dK", memory.dwAvailPhys/1024);
 GetDlgItem(IDC_EDPHYFREE)->SetWindowText(strmem);
 strmem.Format("%dK", memory.dwTotalVirtual/1024);
 GetDlgItem(IDC_EDVIRTOTAL)->SetWindowText(strmem);
 strmem.Format("%dK", memory.dwAvailVirtual/1024);
 GetDlgItem(IDC_EDVIRFREE)->SetWindowText(strmem);
 return TRUE;
}
```

### 举一反三

根据本实例，读者可以：



- 开发内存图形动态显示程序;
- 开发内存报警提示程序。

## 实例 230 监视剪贴板内容

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\06\230

### 实例说明

剪贴板是 Windows 系统中一种主要的数据传输方式,当用户在文字处理过程中选择一段文字,然后选择“复制”命令时,这段文字就被存储到剪贴板中。当用户选择“粘贴”命令时,剪贴板中的文字就会被复制到当前位置。本实例实现查看剪贴板中的内容的功能。运行程序,如果剪贴板中有内容,在列表框中将显示剪贴板中数据的类型,选择列表中的项,不同类型的数据就显示在文本框中,如图 6.24 所示。

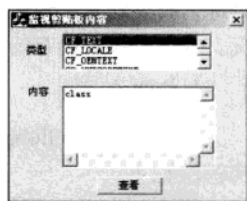


图 6.24 监视剪贴板内容

### 技术要点

本实例主要通过 OpenClipboard、EnumClipboardFormats 和 GetClipboardData 等函数实现,其中 OpenClipboard 函数用于打开剪贴板,没有参数;EnumClipboardFormats 函数用于枚举出剪贴板中所包含数据的所有类型,其语法如下:

```
UINT EnumClipboardFormats(UINT format);
```

参数说明:

- format: 如果要枚举类型,此参数应为 0。
- 返回值: 本函数返回剪贴板中数据的类型。

GetClipboardData 函数是获取剪贴板中不同类型的数据,其语法如下:

```
HANDLE GetClipboardData(UINT uFormat);
```

参数说明:

- uFormat: 剪贴板中数据的类型。
- 返回值: 如果函数执行成功则返回剪贴板的句柄。

### 实现过程

- (1) 新建一个名为 ClipboardView 的对话框 MFC 工程。
- (2) 在对话框上添加两个静态文本控件;添加列表框控件,设置 ID 属性为 IDC\_LISTTYPE,添加成员变量 m\_list;添加文本编辑控件,设置 ID 属性为 IDC\_EDBODY。

- (3) 在头文件 ClipboardViewDlg.h 添加如下变量声明:

```
HWND view;
```

- (4) 在 OnInitDialog 函数中打开剪贴板,代码如下:

```
BOOL CClipboardViewDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 view=SetClipboardViewer();//设置剪贴板视图

 return TRUE;
}
```

- (5) 窗体关闭后,关闭剪贴板,代码如下:

```
void CClipboardViewDlg::OnDestroy()
{
 CDialog::OnDestroy();
 ChangeClipboardChain(view);
}
```

- (6) “查看”按钮的实现函数,该函数用于将剪贴板中数据的类型添加到列表控件中,代码如下:

```
void CClipboardViewDlg::OnView()
{
 m_list.ResetContent();
 if(OpenClipboard())
 {
 UINT i=0;
 while(i<EnumClipboardFormats(i))
 {
 CString str;
 str=GetName(i); //自定义函数，获得类型的名称
 m_list.AddString(str);
 }
 }
}
```

(7) 添加 IDC\_LISTTYPE 控件的 LBN\_SELCHANGE 消息的实现函数，该函数用来获得剪贴板中数据，代码如下：

```
void CClipboardViewDlg::OnSelchangeListtype()
{
 if(IsClipboardFormatAvailable(CF_TEXT))
 {
 HANDLE handle=GetClipboardData(CF_TEXT); //获取文本数据
 char* data=(char*)GlobalLock(handle);
 CString str=data;
 GetDlgItem(IDC_EDBODY)->SetWindowText(str);
 }
}
```

### 举一反三

根据本实例，读者可以：

- 改变桌面背景颜色；
- 设置显示器为节能模式。

## 实例 231

### 利用钩子技术实现键盘监控

这是一个可以提高分析能力的实例

实例位置：光盘\mingrisoft\06\231

### 实例说明

钩子是 WINDOWS 中消息处理机制的一个要点，通过安装各种钩子，应用程序能够设置相应的子例程来监视系统里的消息传递以及在这些消息到达目标窗口程序之前处理它们。钩子的种类很多，每种钩子可以截获并处理相应的消息，如键盘钩子可以截获键盘消息，鼠标钩子可以截获鼠标消息，外壳钩子可以截获启动和关闭应用程序的消息，日志钩子可以监视和记录输入事件。本实例利用钩子实现对键盘的监控。运行程序，用户每按下一个键盘按键，就会弹出对话框来提示用户按下的是什么键，如图 6.25 所示。

### 技术要点

本实例主要通过 SetWindowsHookEx 函数实现，该函数主要用来启动钩子，其语法如下：

```
HHOOK SetWindowsHookEx(int idHook, HOOKPROC lpfn,
 HINSTANCE hMod, DWORD dwThreadId);
```

参数说明：

- idHook：指定钩子的类型，取值如下：
  - WH\_CALLWNDPROC：系统将消息发送到指定窗口之前的钩子。
  - WH\_CALLWNDPROCRET：在窗口中正在处理的消息钩子。
  - WH\_CBT：基于计算机培训的应用程序的钩子。

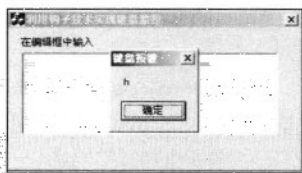


图 6.25 利用钩子技术实现键盘监控

- WH\_DEBUG: 对应系统调试的钩子。
- WH\_FOREGROUNDIDLE: 前台空闲窗口钩子。
- WH\_GETMESSAGE: 接受消息投递钩子。
- WH\_JOURNALPLAYBACK: 回放记录输入消息钩子。
- WH\_JOURNALRECORD: 输入消息记录钩子。
- WH\_KEYBOARD: 键盘消息钩子。
- WH\_MOUSE: 鼠标消息钩子。
- WH\_MSGFILTER: 对话框、消息框、菜单或滚动条输入消息钩子。
- WH\_SHELL: 外壳钩子。
- WH\_SYSMSGFILTER: 系统消息钩子。
- lpfn: 钩子事件触发时调用的回调函数。
- hMod: 包含回调函数的动态链接库文件的句柄。
- dwThreadId: 钩子函数所要监控的应用程序进程号。如果为0则监控所有进程。

## 实现过程

- (1) 新建一个名为KeyboardHook的对话框MFC工程。
- (2) 在对话框上添加文本编辑控件, ID为IDC\_EDUSRINPUT。
- (3) 在KeyboardHookDlg.cpp文件中添加全局钩子句柄及钩子函数的声明, 代码如下:

```
HHOOK hKeyHook;
LRESULT CALLBACK KeyHook(int code, WPARAM wParam, LPARAM lParam);
```

- (4) 在OnInitDialog函数中设置钩子, 代码如下:

```
BOOL CKeyboardHookDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 //设置键盘钩子
 hKeyHook=SetWindowsHookEx(WH_KEYBOARD,(HOOKPROC)KeyHook,
 (HINSTANCE)AfxGetApp()->m_hInstance,0);
 return TRUE;
}
```

- (5) 在关闭窗口的时候卸下钩子, 代码如下:

```
BOOL CKeyboardHookDlg::DestroyWindow()
{
 UnhookWindowsHookEx(hKeyHook);//卸载钩子
 return CDialog::DestroyWindow();
}
```

- (6) 钩子回调函数, 代码如下:

```
LRESULT CALLBACK KeyHook(int code, WPARAM wParam, LPARAM lParam)
{
 char *strkey;
 switch(wParam)//获取消息数据
 {
 case '1':strkey="1";break;
 case '2':strkey="2";break;
 case '3':strkey="3";break;
 case '4':strkey="4";break;
 case '5':strkey="5";break;
 case '6':strkey="6";break;
 case '7':strkey="7";break;
 case '8':strkey="8";break;
 case '9':strkey="9";break;
 case '0':strkey="0";break;
 ...//此处代码省略
 }
 MessageBox(NULL,strkey,"键盘按键",MB_OK);
 return 0;
}
```

## 举一反三

根据本实例，读者可以：

- 利用钩子屏蔽星号获取密码。

## 6.6 程序相关

本节内容包括用列表显示系统正在运行的程序、为程序添加快捷键等实例程序，通过这些实例将简单讲解编程过程中与应用程序相关的一些应用技巧。

## 实例 232

## 用列表显示系统正在运行的程序

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\06\232

## 实例说明

本实例完成枚举系统正在运行的程序，运行程序后，系统中正在运行的程序就显示在列表中，如图 6.26 所示，列表中所列出的程序和任务管理器“进程”选项卡所列出的程序基本一致。

## 技术要点

本实例主要应用了 CreateToolhelp32Snapshot、Process32First 和 Process32Next 等函数，实现了枚举系统正在运行的程序的功能。

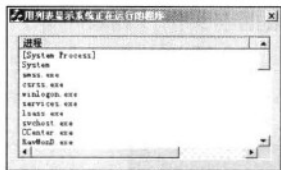


图 6.26 用列表显示系统正在运行的程序

下面重点介绍 CreateToolhelp32Snapshot 函数。

CreateToolhelp32Snapshot 函数是对当前系统中进程生成快照，其语法如下：

HANDLE WINAPI CreateToolhelp32Snapshot(DWORD dwFlags, DWORD th32ProcessID);

参数说明：

- dwFlags：快照的类型，取值如下。
  - TH32CS\_INHERIT：快照句柄将被继承。
  - TH32CS\_SNAPALL：相当于 TH32CS\_SNAPHEAPLIST、TH32CS\_SNAPMODULE、TH32CS\_SNAPPROCESS 和 TH32CS\_SNAPTHREAD 一起调用。
  - TH32CS\_SNAPHEAPLIST：指定进程堆列表的快照。
  - TH32CS\_SNAPMODULE：指定进程模块列表的快照。
  - TH32CS\_SNAPPROCESS：进程的快照。
  - TH32CS\_SNAPTHREAD：线程快照。
- th32ProcessID：进程的 ID 值。

## 实现过程

- (1) 新建一个名为 ProcessList 的对话框 MFC 工程。
- (2) 在对话框上添加列表视图控件，添加成员变量 m\_list。
- (3) 在 ProcessListDlg.cpp 文件中加入头文件引用：  
#include "thelp32.h"
- (4) 在 OnInitDialog 函数中通过对系统进行快照将系统的进程显示出来，代码如下：  

```

BOOL CProcessListDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_list.SetExtendedStyle(LVS_EX_GRIDLINES);
}

```



```

m_list.InsertColumn(0,"进程",LVCFMT_LEFT,300,0);
m_list.InsertColumn(1,"ID",LVCFMT_CENTER,75,1);
m_list.InsertColumn(2,"父进程",LVCFMT_CENTER,75,2);
HANDLE toolhelp=CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
if(toolhelp==NULL)return FALSE;
m_list.SetRedraw(FALSE);
PROCESSENTRY32 processinfo;
int i=0;
CString str;
BOOL start=Process32First(toolhelp,&processinfo);//获取第一个进程
while(start)
{
 m_list.InsertItem(i,"");
 m_list.SetItemText(i,0,processinfo.szExeFile);
 str.Format("%08x",processinfo.th32ProcessID);
 m_list.SetItemText(i,1,str);
 str.Format("%08x",processinfo.th32ParentProcessID);
 m_list.SetItemText(i,2,str);
 start=Process32Next(toolhelp,&processinfo);//获取下一个进程
 i++;
}
m_list.SetRedraw(TRUE);
return TRUE;
}

```

### 举一反三

根据本实例，读者可以：

- 在进程中关闭系统正在运行的程序；
- 分类枚举系统中正在运行的程序。

## 实例 233 为程序添加快捷方式

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\06\233

### 实例说明

一般软件的安装过程中，打包程序会在安装将要结束的时候，在系统开始菜单或桌面为程序添加快捷方式，本实例实现在应用程序中为其他程序添加快捷方式。运行程序，单击“浏览”按钮选择要添加快捷方式的程序，如图 6.27 所示，单击“创建桌面快捷方式”按钮，程序会在桌面生成快捷方式，如图 6.28 所示，单击“创建菜单快捷方式”程序会在开始菜单中生成快捷方式。

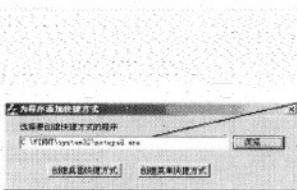


图 6.27 为程序添加快捷方式

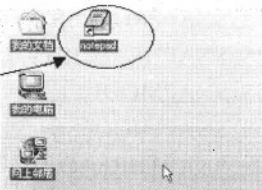


图 6.28 创建桌面快捷方式后的结果

### 技术要点

本实例主要通过 IShellLink 接口来实现添加快捷方式。快捷方式分为桌面快捷方式和开始菜单快捷方式，实现方法主要是找到这两个系统文件夹，然后在这两个文件夹内添加扩展名为 lnk 的文件，最后使用 IShellLink 接口设置文件的属性。要找到两个系统文件夹，需要通过 SHGetSpecialFolderLocation 函数实现，桌面文件夹使用参数 CSIDL\_DESKTOP，开始菜单程序组使用参数 CSIDL\_PROGRAMS，该函数的语法如下：

```

WINHELLAPI HRESULT WINAPI SHGetSpecialFolderLocation(HWND hwndOwner,
int nFolder,LPTITEMIDLIST *ppidl);

```

参数说明:

- hwndOwner: 应用程序句柄。
- nFolder: 特殊文件夹的标识。
- \*ppidl: 说明文件夹位置的指针。

找到两个系统文件夹后,通过 IShellLink 的 SetPath 方法设置需要创建快捷方式的应用程序的路径,通过 IPersistFile 的 Save 方法来设置快捷方式的路径。

另外,为了通知系统相应的设置应该生效,实例中还使用了 SHChangeNotify 函数。

## 实现过程

(1) 新建一个名为 AddShortcut 的对话框 MFC 工程。

(2) 在对话框上添加 1 个静态文本控件;添加 3 个按钮控件,设置 ID 属性分别为 IDC\_ADD、IDC\_DESKTOP 和 IDC\_GROUP,设置 Caption 属性分别为“浏览...”、“创建桌面快捷方式”和“创建菜单快捷方式”。

(3) 在实现文件 AddShortcutDlg.cpp 中加入下面语句:

```
#include "objidl.h"
```

(4) 在头文件 AddShortcutDlg.h 中加入下面语句:

```
CString pathname;
CString strname;
```

(5) “浏览...”按钮的实现函数,该函数用于打开要建快捷方式的应用程序,代码如下:

```
void CAddShortcutDlg::OnAdd()
{
 CFileDialog log(FALSE, "可执行文件", "*.EXE", OFN_HIDEREADONLY, "可执行程序|*.exe||", NULL);
 if(log.DoModal() != IDOK)
 {
 pathname=log.GetPathName();//获取路径
 strname=log.GetFileName();//获取文件名
 GetDlgItem(IDC_EDADD)->SetWindowText(pathname);
 }
}
```

(6) 按钮“创建桌面快捷方式”的实现函数,该函数用于创建 LNK 文件从而创建桌面快捷方式,代码如下:

```
void CAddShortcutDlg::OnDesktop()
{
 if(strname.IsEmpty())return;
 IShellLink *link;
 IPersistFile *file;
 HRESULT res=::CoCreateInstance(CLSID_ShellLink,NULL,
 CLSCTX_INPROC_SERVER,IID_IShellLink,(void **)&link);
 if(FAILED(res))//创建接口实例
 return;
 link->SetPath(pathname);//设置路径
 res=link->QueryInterface(IID_IPersistFile,(void **)&file);//查询接口
 if(FAILED(res))
 return;
 WORD wsz[MAX_PATH];
 CString linkname;
 LPITEMIDLIST pid;
 char path[MAX_PATH];
 ::SHGetSpecialFolderLocation(NULL,CSIDL_DESKTOP,&pid);//获取桌面PID
 ::SHGetPathFromIDList(pid,path);

 CString name;
 int pos=strname.Find(".");
 name=strname.Left(pos);
 linkname.Format("%s\\%s.lnk",path,name);
 MultiByteToWideChar(CP_ACP,0,linkname,-1,wsz,MAX_PATH);
 file->Save(wsz,STGM_READWRITE);
 file->Release();
 link->Release();
 ::SHChangeNotify(SHCNE_CREATE|SHCNE_INTERRUPT,SHCNF_FLUSH|SHCNF_PATH,linkname,0);
 ::SHChangeNotify(SHCNE_UPDATEDIR|SHCNE_INTERRUPT,
 SHCNF_FLUSH|SHCNF_PATH,path,0);
}
```

(7) 按钮“创建菜单快捷方式”的实现函数,该函数通过创建 LNK 文件从而创建菜单快

捷方式，代码如下：

```
void CAddShortcutDlg::OnGroup()
{
 if(strname.IsEmpty())return;
 IShellLink *link;
 IPersistFile *file;
 HRESULT res=::CoCreateInstance(CLSID_ShellLink,NULL,
 CLSCTX_INPROC_SERVER,IID_IShellLink,(void **)&link);
 if(FAILED(res))//创建接口实例
 return;
 link->SetPath(pathname);//设置文件路径
 res=link->QueryInterface(IID_IPersistFile,(void **)&file);
 if(FAILED(res))
 return;
 WORD wsz[MAX_PATH];
 CString linkname;
 LPITEMIDLIST pid;
 char path[MAX_PATH];
 ::SHGetSpecialFolderLocation(NULL,CSIDL_PROGRAMS,&pid);//获取程序组PID
 ::SHGetPathFromIDList(pid,path);

 CString name;
 int pos=strname.Find(".");
 name=strname.Left(pos);
 linkname.Format("%s\\%s.lnk",path,name);
 MultiByteToWideChar(CP_ACP,0,linkname,-1,wsz,MAX_PATH);
 file->Save(wsz,STGM_READWRITE);
 file->Release();
 link->Release();
 ::SHChangeNotify(SHCNE_CREATE|SHCNE_INTERRUPT,SHCNF_FLUSH|SHCNF_PATH,
 linkname,0); //通知系统更新变化
 ::SHChangeNotify(SHCNE_UPDATEDIR|SHCNE_INTERRUPT,
 SHCNF_FLUSH|SHCNF_PATH,path,0);
}
```

### 举一反三

根据本实例，读者可以：

- 删除快捷方式；
- 在开始菜单生成快捷方式组。

## 实例 234

### 设置其他程序中编辑框内的文本

这是一个可以提高基础性能的实例

实例位置：光盘\mingrisoft\06\234

### 实例说明

在开发应用程序时，有时需要设置其他进程中对话框的文本。众所周知，Windows 操作系统对进程的保护非常严格，如何设置其他进程中对话框的文本呢？本例实现了该功能，效果如图 6.29、图 6.30 所示。

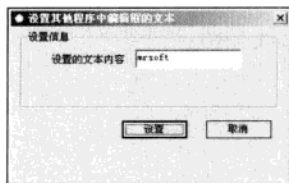


图 6.29 主控窗口

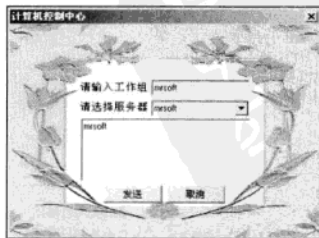


图 6.30 被控窗口

### 技术要点

要设置其他进程中对话框的文本，首先需要获取进程中对话框句柄，然后通过对话框句柄获取其子窗口（编辑框）句柄，最后通过子窗口句柄设置子窗口文本。

在程序中,可以通过 EnumWindows 函数列举所有的对话框,该函数语法如下:

```
BOOL EnumWindows(WNDENUMPROC lpEnumFunc, LPARAM lParam);
```

参数说明: lpEnumFunc 是回调函数指针,其格式如下:

```
BOOL CALLBACK EnumWindowsProc(HWND hwnd,LPARAM lParam);
```

其中 hwnd 表示列举的对话框句柄, lParam 表示由 EnumWindows 函数传递的附加信息,也就是 EnumWindows 函数中为 lParam 参数设置的参数值。

可以通过 EnumChildWindows 函数根据对话框句柄获得对话框中子窗口的句柄。该函数语法如下:

```
BOOL EnumChildWindows(HWND hWndParent, WNDENUMPROC lpEnumFunc, LPARAM lParam);
```

参数说明: hWndParent 表示对话框句柄, lpEnumFunc 表示一个回调函数指针, lParam 表示向回调函数中传递的参数。

在设置子窗口文本时,不能通过 SetWindowText 函数实现,该函数只能设置同一个进程中子窗口的文本,而应通过发送 WM\_SETTEXT 消息来实现,消息参数 lParam 标识了文本内容。

## 实现过程

(1) 新建一个基于对话框的应用程序。在对话框中添加静态文本、文本编辑和按钮控件。

(2) 添加全局函数 EnumWindowsProc,用于列举所有的对话框句柄,代码如下:

```
BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM lParam)
{
 CString str;
 GetWindowText(hwnd,str.GetBuffer(0),100);
 if (str=="计算机控制中心")
 {
 EnumChildWindows(hwnd,EnumChildWindowsProc,lParam);
 }
 return TRUE;
}
```

(3) 添加全局函数 EnumChildWindowsProc,用于列举某个对话框中所有的子窗口句柄,代码如下:

```
BOOL CALLBACK EnumChildWindowsProc(HWND hwnd, LPARAM lParam)
{
 CString str;
 ::SendMessage(hwnd,WM_GETTEXT,100,(LPARAM)str.GetBuffer(0));
 if (str=="")
 {
 ::SendMessage(hwnd,WM_SETTEXT,0,lParam);
 }
 return TRUE;
}
```

(4) 处理“设置”按钮的单击事件,设置其他进程中子窗口的文本,代码如下:

```
void CGetEditDlg::OnOK()
{
 CString str;
 m_Content.GetWindowText(str);
 EnumWindows(EnumWindowsProc,(LPARAM)str.GetBuffer(0));
}
```

## 举一反三

根据本实例的设计思路和一些技术要点,读者还可以:

- 获取其他进程中编辑框内的文本。

### 实例 235

### 执行一个外部程序直到其结束

本实例是一个人性化的实例

实例位置:光盘\mingrisoft\06\235

## 实例说明

一个大的应用程序通常由多个模块组成,每个模块执行不同的功能,由主模块统一调动。通常情况下,在调用一个子模块时,主模块还能继续响应用户操作,但有时需要限制此种情况。例



如在子模块运行时,禁止主模块执行操作。本例实现了该功能,当主窗口运行时,如图 6.31 所示,单击“注册表”按钮,打开“注册表编辑器”窗口,如图 6.32 所示,此时主窗口将不可用。

## 技术要点

本例中需要解决两个问题,一是如何调用子模块,二是在子模块运行时,如何禁止响应主模块操作。

对于问题一,可以通过调用 `CreateProcess` 函数实现,该函数语法如下:

```
BOOL CreateProcess(LPCTSTR lpApplicationName, LPTSTR lpCommandLine, LPSECURITY_ATTRIBUTES lpProcessAttributes, LPSECURITY_ATTRIBUTES lpThreadAttributes, BOOL bInheritHandles, DWORD dwCreationFlags, LPVOID lpEnvironment, LPCTSTR lpCurrentDirectory, LPSTARTUPINFO lpStartupInfo, LPPROCESS_INFORMATION lpProcessInformation);
```

参数说明: `lpApplicationName` 标识应用程序的名称, `lpCommandLine` 标识应用程序命令行信息, `lpProcessAttributes` 和 `lpThreadAttributes` 是一个 `SECURITY_ATTRIBUTES` 结构指针,分别标识进程安全属性和线程安全属性, `bInheritHandles` 标识新进程是否继承被调用进程的句柄, `dwCreationFlags` 是一组标识,用于指定进程的创建标识, `lpEnvironment` 用于指定进程的环境变量, `lpCurrentDirectory` 用于为进程指定当前工作目录, `lpStartupInfo` 是 `STARTUPINFO` 结构指针,记录进程的启动信息, `lpProcessInformation` 是 `PROCESS_INFORMATION` 结构指针,记录进程句柄、线程句柄、进程 ID、线程 ID 等信息。

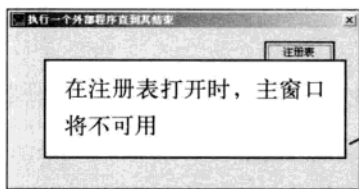


图 6.31 主窗口

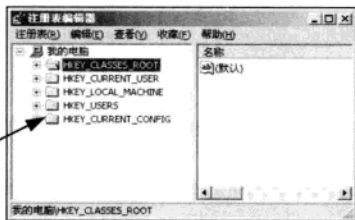


图 6.32 注册表编辑器窗口

对于问题二,可以通过调用 `WaitForSingleObject` 函数实现,该函数在某个对象处于有信号状态或超时时返回,其语法如下:

```
DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds);
```

参数说明: `hHandle` 标识对象句柄,本例应为进程句柄。 `dwMilliseconds` 标识等待的毫秒数,如果为 `INFINITE`,标识一直等待,直到进程有信号为止。

## 实现过程

- (1) 新建一个基于对话框的应用程序。在对话框中添加按钮控件。
- (2) 主要程序代码如下:

```
void CWaitProcessDlg::OnExecute()
{
 STARTUPINFO strinfo;
 PROCESS_INFORMATION processinfo;

 memset(&strinfo, 0, sizeof(strinfo));
 strinfo.cb = sizeof(strinfo);
 //创建进程
 BOOL ret = CreateProcess(NULL, "regedit.exe",
 NULL, NULL, TRUE, DETACHED_PROCESS, NULL, NULL, &strinfo, &processinfo);

 if (ret)
 {
 //进程等待
 WaitForSingleObject(processinfo.hProcess, INFINITE);
 }
}
```

## 举一反三

根据本实例，读者可以：

- 在程序中调用查询分析器执行 SQL 脚本。

## 实例 236

## 调用具有参数的可执行程序

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\06\236

## 实例说明

在设计应用程序时，经常将某一类操作封装在一个功能模块中，由其他程序调用。但是，在调用功能模块时如何确定执行哪一项操作呢？例如功能模块中包含了“备份数据库”、“分离数据库”和“停止服务”等功能，在调用功能模块时，如何确定具体执行哪个功能呢？本例实现了一个调用具有参数的可执行程序。运行程序，如图 6.33 所示。

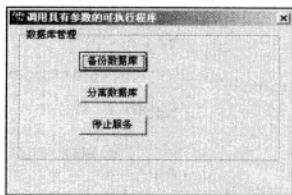


图 6.33 调用具有参数的可执行程序

## 技术要点

在设计功能模块时，为了区分调用进程具体执行哪一个项操作，需要为其设置参数。可以利用命令行信息实现。在主程序调用 CreateProcess 函数创建进程时，指定命令行信息。在功能模块中调用 GetCommandLine 方法获得命令行信息，并根据不同的命令行信息，执行不同的功能。具体代码可参见实现过程。

## 实现过程

功能模块设计步骤。

- 新建一个基于对话框的应用程序。
- 定义一个全局变量，用于记录命令行信息，代码如下：

```
CString cmd;
```

- 在应用程序初始化时获取命令行信息，代码如下：

```
cmd = GetCommandLine(); //获得命令行信息
```

- 在对话框初始化时，解析命令行信息。根据不同的信息执行相应的操作，代码如下：

```
CString str;
str = cmd.Left(8);
```

```
CString sql;
```

```
if (str=="停止服务")
```

```
{
```

```
 sql.Format("SHUTDOWN");
```

```
 try
```

```
 {
```

```
 pCon->Execute((_bstr_t)sql,NULL,0); //执行SQL语句
```

```
 }
```

```
 catch(...)
```

```
 {
```

```
 MessageBox("操作失败");
```

```
 }
```

```
}
```

```
else if (str=="备份数据")
```

```
{
```

```
 //获取数据库名称
```

```
 CString database;
```

```
 int pos1 = cmd.Find("*");
```

```
 int pos2 = cmd.Find("*",pos1+1);
```

```
 database = cmd.Mid(pos1+1,pos2-pos1-1);
```

```
 //获取备份文件
```

```
 CString path = cmd.Right(cmd.GetLength()-pos2-1);
```

```

sql.Format("backup database %s to disk = '%s'", database, path);
try
{
 pCon->Execute((_bstr_t)sql, NULL, 0); //执行SQL语句
}
catch(...)
{
 MessageBox("操作失败");
}
}
else if (str == "分离数据")
{
 //获取数据库名称
 CString database;
 int pos1 = cmd.Find("**");
 int pos2 = cmd.Find("**", pos1+1);

 database = cmd.Mid(pos1+1);
 sql.Format("sp_detach_db '%s'", database);
 try
 {
 pCon->Execute((_bstr_t)sql, NULL, 0);
 }
 catch(...)
 {
 MessageBox("操作失败");
 }
}
}

```

主程序设计步骤如下。

- (1) 创建一个基于对话框的应用程序。在对话框中添加按钮控件。
- (2) 处理“备份数据库”按钮的单击事件，调用功能模块备份数据库，代码如下：

```

//备份数据库
void CParamExeDlg::OnBackup()
{
 SECURITY_ATTRIBUTES ProSec, ThreadSec;

 ProSec.bInheritHandle = TRUE;
 ProSec.nLength = sizeof(ProSec);

 ThreadSec.bInheritHandle = TRUE;
 ThreadSec.nLength = sizeof(ThreadSec);

 STARTUPINFO StrInfo;
 memset(&StrInfo, 0, sizeof(StrInfo));
 StrInfo.cb = sizeof(StrInfo);
 PROCESS_INFORMATION ProclInfo;

 CreateProcess("DataManage.exe", "备份数据*master*c:\\master.bak",
 NULL, NULL, TRUE, NORMAL_PRIORITY_CLASS|CREATE_NEW_CONSOLE,
 NULL, NULL, &StrInfo, &ProclInfo);
}

```

- (3) 处理“分离数据库”按钮的单击事件，调用功能模块分离数据库，代码如下：

```

//分离数据库
void CParamExeDlg::OnDetach()
{
 SECURITY_ATTRIBUTES ProSec, ThreadSec;

 ProSec.bInheritHandle = TRUE;
 ProSec.nLength = sizeof(ProSec);

 ThreadSec.bInheritHandle = TRUE;
 ThreadSec.nLength = sizeof(ThreadSec);

 STARTUPINFO StrInfo;
 memset(&StrInfo, 0, sizeof(StrInfo));
 StrInfo.cb = sizeof(StrInfo);
 PROCESS_INFORMATION ProclInfo;

 CreateProcess("DataManage.exe", "分离数据*pubs", NULL, NULL, TRUE,
 NORMAL_PRIORITY_CLASS|CREATE_NEW_CONSOLE, NULL,
 NULL, &StrInfo, &ProclInfo);
}

```

### 举一反三

根据本实例，读者可以：

灵活设计功能模块。

## 实例 237 编写控制面板小应用程序

本实例是一个人性化的实例

实例位置:光盘\mingrisoft\06\237

### 实例说明

在 Windows 的控制面板中有许多应用程序,这些应用程序被称之为控制面板小程序,那么是否可以创建这种控制面板小程序呢?本例就是使用 Visual C++来设计控制面板小应用程序。运行本实例,将生成的 DLL 文件修改为 CPL 文件,并将其复制到系统目录下,打开控制面板。运行程序,如图 6.34 所示。

### 技术要点

本示例中设计控制面板小应用程序时,主要用到了 DLL 的创建,下面对本实例中用到的关键技术进行详细讲解。

DLL 的创建步骤如下。

- (1) 单击 File/New 菜单项,打开 New 窗口,选择 Projects 选项卡,如图 6.35 所示。
- (2) 在列表中选择 MFC AppWizard(dll)选项,表示创建一个动态链接库,在 Project name 编辑框中输入工程名称。单击“OK”按钮进入向导窗口,如图 6.36 所示。



图 6.34 编写控制面板小应用程序

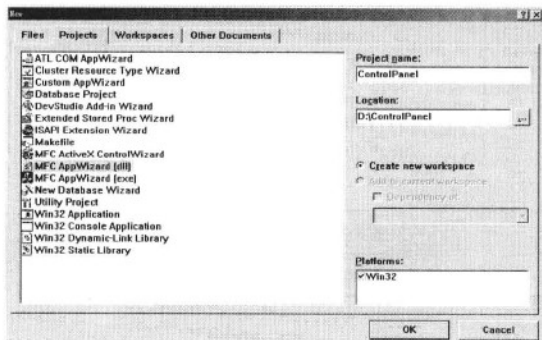


图 6.35 New 窗口

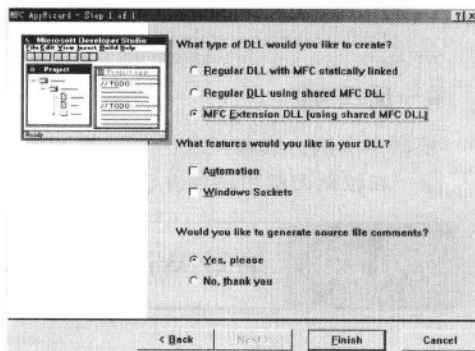


图 6.36 向导窗口

- (3) 选择“MFC Extension DLL(using shared MFC DLL)”单选按钮,表示创建一个扩展 DLL,单击“Finish”按钮完成 DLL 的创建。

### 实现过程

- (1) 按照技术要点中的步骤创建一个 DLL。
- (2) 向工程导入一个图标资源,并在字符串编辑器中编辑两个字符串资源,如图 6.37 所示。
- (3) 添加回调函数 CPIApplet,在该函数中设置显示图标、名称、提示文本及双击功能,代码如下:

```
long CALLBACK CPIApplet(HWND hParent,UINT msg,long wParam,long lParam)
{
 switch (msg)
```

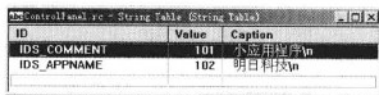


图 6.37 字符串编辑器



```

{
case CPL_INIT:
{
return TRUE;
}
case CPL_GETCOUNT:
{
return 1;
}

case CPL_INQUIRE:
{
LPCPLINFO pInfo = (LPCPLINFO)lParam;
pInfo->idlIcon = IDI_ICON1; //显示图标
pInfo->idlInfo = IDS_COMMENT; //提示文本
pInfo->idlName = IDS_APPNAME; //显示名称
break;
}

case CPL_DBLCLK:
{
MessageBox(0, "明日科技欢迎您!", "提示", 64); //双击控制面板图标时弹出消息框
break;
}

case CPL_STOP:
case CPL_EXIT:
break;
}
return 0;
}

```

(4) 向 DEF 文件中加入回调函数 CPIApplet 的声明, 其实现代码如下:

; ControlPanel.def: Declares the module parameters for the DLL.

```

LIBRARY "ControlPanel"
DESCRIPTION 'ControlPanel Windows Dynamic Link Library'

EXPORTS
; Explicit exports can go here
CPIApplet //回调函数声明

```

## 举一反三

根据本实例, 读者可以:

- 在控制面板中添加自己的服务。

## 实例 238 编写 Windows 服务

本实例是一个人性化的实例

实例位置: 光盘\mingrisoft\06\238

## 实例说明

Windows 服务能够执行许多幕后的操作, 许多大型的应用软件, 尤其是数据库管理软件, 都包含有许多 Windows 服务。本例中笔者设计了一个简单的 Windows 服务, 在服务启动后, 能够通过程序控制服务执行相应的动作。运行程序, 如图 6.38 所示。

## 技术要点

在本实例中服务的自动启动是通过 CreateService 函数来完成的, 该函数的功能是可以创建一个服务并启动。参数原型如下:

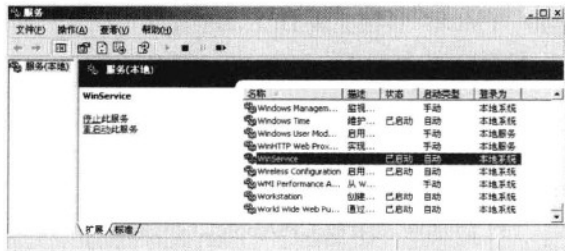


图 6.38 编写 Windows 服务

```
SC_HANDLE CreateService(
 SC_HANDLE hSCManager,
 LPCTSTR lpServiceName,
 LPCTSTR lpDisplayName,
 DWORD dwDesiredAccess,
 DWORD dwServiceType,
 DWORD dwStartType,
 DWORD dwErrorControl,
 LPCTSTR lpBinaryPathName,
 LPCTSTR lpLoadOrderGroup,
 LPDWORD lpdwTagId,
 LPCTSTR lpDependencies,
 LPCTSTR lpServiceStartName,
 LPCTSTR lpPassword
);
```

函数说明如下。

- hSCManager: 服务控制管理器数据库。
- lpServiceName: 后台服务器名称。
- lpDisplayName: 在服务器控制面板中显示的服务器名称。
- dwDesiredAccess: 可访问权限。
- dwServiceType: 服务类型。
- dwStartType: 启动类型。
- dwErrorControl: 服务启动错误信息。
- lpBinaryPathName: 服务文件路径。
- lpLoadOrderGroup: 服务分组信息。
- lpdwTagId: 分组 ID。
- lpDependencies: 共享服务名称列表。
- lpServiceStartName: 访问此服务的用户名。
- lpPassword: 访问此服务的密码。

## 实现过程

- (1) 创建一个 ATL Com 工程，工程类型为 Service(exe)。
- (2) 然后修改 CServiceModule 的 Install 方法，使服务能够自动运行。
- (3) 向 CServiceModule 类中添加一个自定义的方法 BackupData，用于实现文件的复制。
- (4) 在 CServiceModule 的 Handler 方法中根据参数调用 BackupData 方法。
- (5) 主要程序代码如下：

```
#define CM_BAKUPDATABASE 129
inline BOOL CServiceModule::Install()
{
 if (IsInstalled())
 return TRUE;
 //打开服务管理器
 SC_HANDLE hSCM = ::OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
 if (hSCM == NULL)
 {
 MessageBox(NULL, _T("Couldn't open service manager"), m_szServiceName, MB_OK);
 return FALSE;
 }

 TCHAR szFilePath[_MAX_PATH];
 ::GetModuleFileName(NULL, szFilePath, _MAX_PATH);

 //服务自动启动
 SC_HANDLE hService = ::CreateService(
 hSCM, m_szServiceName, m_szServiceName,
 SERVICE_ALL_ACCESS, SERVICE_WIN32_OWN_PROCESS,
 SERVICE_AUTO_START, SERVICE_ERROR_NORMAL,
```

```

 szFilePath, NULL, NULL, NULL, NULL, NULL);

 if (hService == NULL)
 {
 ::CloseServiceHandle(hSCM);
 MessageBox(NULL, _T("Couldn't create service"), m_szServiceName, MB_OK);
 return FALSE;
 }

 ::CloseServiceHandle(hService);
 ::CloseServiceHandle(hSCM);
 return TRUE;
}

LRESULT CServiceModule::BackupData()
{
 CopyFile("c:\\Book.mdb", "d:\\Book.mdb", true); //复制文件
 return 0;
}

inline void CServiceModule::Handler(DWORD dwOpcode)
{
 switch (dwOpcode)
 {
 case SERVICE_CONTROL_STOP://停止服务
 SetServiceStatus(SERVICE_STOP_PENDING);
 PostThreadMessage(dwThreadId, WM_QUIT, 0, 0);
 break;
 case SERVICE_CONTROL_PAUSE://暂停服务
 break;
 case SERVICE_CONTROL_CONTINUE://继续
 break;
 case SERVICE_CONTROL_INTERROGATE:
 break;
 case SERVICE_CONTROL_SHUTDOWN:
 break;
 case CM_BAKUPDATABASE: //129
 BackupData();
 break;
 default:
 LogEvent(_T("Bad service request"));
 }
}

void CServiceModule::Run()
{
 _Module.dwThreadId = GetCurrentThreadId();

 HRESULT hr = CoInitialize(NULL);
 _ASSERT(SUCCEEDED(hr));
 CSecurityDescriptor sd;
 sd.InitializeFromThreadToken();
 hr = CoInitializeSecurity(sd, -1, NULL, NULL,
 RPC_C_AUTHN_LEVEL_PKT, RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE, NULL);
 _ASSERT(SUCCEEDED(hr));

 hr = _Module.RegisterClassObjects(CLSCTX_LOCAL_SERVER | CLSCTX_REMOTE_SERVER, REGCLS_MULTIPLEUSE);
 _ASSERT(SUCCEEDED(hr));

 LogEvent(_T("Service started"));
 if (m_bService)
 SetServiceStatus(SERVICE_RUNNING); //运行服务

 MSG msg;
 while (GetMessage(&msg, 0, 0, 0))
 {
 DispatchMessage(&msg);
 }

 _Module.RevokeClassObjects();

 CoUninitialize();
}

```

(6) 运行程序, 在 Windows 服务中会发现编写的服务, 为了说明该服务的作用, 本例

中还包含了一个演示程序。它调用服务中的 BackupData 方法实现了文件的复制。演示程序代码如下：

```
void CServerDlg::OnOK()
{
 SC_HANDLE hManage = OpenSCManager(NULL, NULL,
 SC_MANAGER_ENUMERATE_SERVICE|GENERIC_EXECUTE);

 SC_HANDLE hService = OpenService(hManage, "WinService", SERVICE_ALL_ACCESS);
 SERVICE_STATUS State;
 //129是服务中Handler方法的参数，调用BackupData方法
 BOOL ret = ControlService(hService, 129, &State);
 CloseServiceHandle(hService);
 CloseServiceHandle(hManage);
}
```

### 举一反三

根据本实例，读者可以：

- 设计数据传输服务。

## 实例 239 阻止程序重复运行

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\06\239

### 实例说明

有些应用程序在操作系统中不允许同时运行两个或多个实例，特别是服务性的应用程序，所以需要程序设计人员阻止程序的重复运行。运行程序，如图 6.39 所示。

### 技术要点

本实例实现的阻止程序的重复运行是通过互斥内核对象实现的，在程序所在的工程初始化时创建一个互斥对象，工程退出时释放这个互斥对象。CreateMutex 函数和 ReleaseMutex 函数就是用来创建和释放互斥对象的。



图 6.39 阻止程序重复运行

### 实现过程

- 新建一个基于对话框的应用程序。
- 在工程初始化方法 InitInstance 中创建互斥对象，代码如下：

```
hmutex=CreateMutex(NULL,TRUE,"NoRepeat");//创建互斥对象
if(GetLastError()==ERROR_ALREADY_EXISTS)
{
 ::MessageBox(NULL,"程序已经运行","提示",MB_OK);
 return FALSE;
}
```

- 在工程退出方法 ExitInstance 中释放互斥对象，代码如下：

```
int CNoRepeatApp::ExitInstance()
{
 ReleaseMutex(hmutex);//释放互斥对象
 return CWinApp::ExitInstance();
}
```

### 举一反三

根据本实例，读者可以：

- 通过 FindWindow 函数阻止程序重复运行。



## 6.7 线程同步

在开发多线程应用程序时,线程同步是必须考虑的问题。本节通过几个实例介绍线程同步的方法。

### 实例 240

### 利用事件对象实现线程同步

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\08\240

#### 实例说明

本实例主要通过事件对象来实现线程的同步。运行程序,单击按钮“不使用事件同步启动两个线程向编辑框中写入不同字符”,启动两个没有使用任何同步对象的线程向编辑框中写入字符 A 和 B,可以看到的现象是字符 A 和字符 B 交叉在编辑框中显示,单击按钮“使用事件同步启动两个线程向编辑框中写入不同字符”,启动两个线程,一个线程向编辑框写入字符 A,并且在任务完成时设置了事件对象,另一个线程向编辑框写入字符 B,并且等待事件对象,如果没有事件对象被重置该线程会一直等待,如图 6.40 所示。

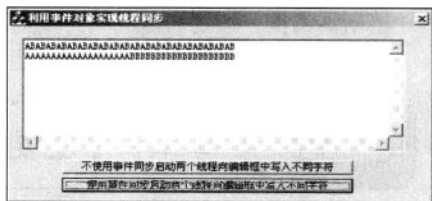


图 6.40 利用事件对象实现线程同步

#### 技术要点

事件对象可以用两种方式来实现,一种是使用 MFC 对象,一种是不使用 MFC 对象,本实例使用了 MFC 的事件对象 CEvent。如果不使用 MFC 对象就需要使用 CreateEvent 函数来创建一个事件对象句柄,然后通过 WaitForSingleObject 等待事件对象和 ResetEvent 重置事件对象两个函数一起来实现线程的同步,本实例在第一个线程任务将要结束的时候重置事件对象,在第二个线程开始就去等待事件对象,直到有事件对象被重置才开始执行任务。重置事件使用 SetEvent 方法,等待事件对象使用 Lock 方法。

#### 实现过程

- (1) 新建一个名为 EventSynch 的对话框 MFC 工程。
- (2) 在对话框上添加一个文本编辑控件,添加成员变量 m\_result;添加两个按钮控件,设置 ID 属性分别为 IDC\_BTSTARTONE 和 IDC\_BTSTARTTWO,设置 Caption 属性分别为“不使用事件同步启动两个线程向编辑框中写入不同字符”和“使用事件同步启动两个线程向编辑框中写入不同字符”。

- (3) 在 StdAfx.h 中添加下面语句:

```
#include <afxcmn.h>
```

- (4) 在 EventSynchDlg.cpp 文件中加入如下全局变量声明:

```
CEvent cEvent;
```

- (5) 全局线程函数,该函数用于向文本框写入字符 A,代码如下:

```
static UINT thread1(LPVOID pParam)
{
 CEdit *p=(CEdit*)pParam;
 char buf[MAX_PATH];
 for(int i=0;i<20;i++)
 {
 ::SendMessage(p->GetSafeHwnd(),WM_GETTEXT,MAX_PATH,(LPARAM)buf);
 strcat(buf,"A");
 ::SendMessage(p->GetSafeHwnd(),WM_SETTEXT,0,(LPARAM)buf);
 }
}
```

```
Sleep(200);
}
return 0;
}
```

(6) 全局线程函数，该函数用于向文本框写入字符 B，代码如下：

```
static UINT thread2(LPVOID pParam)
{
 CEdit *p=(CEdit*)pParam;
 char buf[MAX_PATH];
 for(int i=0;i<20;i++)
 {
 ::SendMessage(p->GetSafeHwnd(),WM_GETTEXT,MAX_PATH,(LPARAM)buf);
 strcat(buf,"B");
 ::SendMessage(p->GetSafeHwnd(),WM_SETTEXT,0,(LPARAM)buf);
 Sleep(200);
 }
 return 0;
}
```

(7) 全局线程函数，该函数用于向文本框写入字符 A，实现对 CEvent 对象的设置，代码如下：

```
static UINT thread3(LPVOID pParam)
{
 CEdit *p=(CEdit*)pParam;
 char buf[MAX_PATH];
 for(int i=0;i<20;i++)
 {
 ::SendMessage(p->GetSafeHwnd(),WM_GETTEXT,MAX_PATH,(LPARAM)buf);
 strcat(buf,"A");
 ::SendMessage(p->GetSafeHwnd(),WM_SETTEXT,0,(LPARAM)buf);
 Sleep(200);
 }
 cEvent.SetEvent();
 return 0;
}
```

(8) 全局线程函数，该函数用于向文本框写入字符 B，通过 Lock 方法等待 CEvent 对象被重置，代码如下：

```
static UINT thread4(LPVOID pParam)
{
 CEdit *p=(CEdit*)pParam;
 char buf[MAX_PATH];
 cEvent.Lock();
 for(int i=0;i<20;i++)
 {
 ::SendMessage(p->GetSafeHwnd(),WM_GETTEXT,MAX_PATH,(LPARAM)buf);
 strcat(buf,"B");
 ::SendMessage(p->GetSafeHwnd(),WM_SETTEXT,0,(LPARAM)buf);
 Sleep(200);
 }
 cEvent.SetEvent();
 return 0;
}
```

(9) “不使用事件同步启动两个线程向编辑框中写入不同字符”按钮的实现函数，该函数用于启动线程函数 thread1 和 thread2，代码如下：

```
void CEventSynchDlg::OnStartOne()
{
 AfxBeginThread(thread1,&m_result);
 AfxBeginThread(thread2,&m_result);
}
```

(10) “利用事件对象实现线程同步”按钮的实现函数，该函数用于启动线程函数 thread3 和 thread4，代码如下：

```
void CEventSynchDlg::OnStartTwo()
{
 AfxBeginThread(thread3,&m_result);
 AfxBeginThread(thread4,&m_result);
}
```

## 举一反三

根据本实例，读者可以：

- 利用事件对象实现多线程下载工具。

## 实例 241

## 利用互斥对象实现线程同步

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\08\241

## 实例说明

本实例通过互斥对象实现线程的同步。运行程序，两个按钮分别启动一个线程向编辑框中写入字符，选中复选框可以实现当一个线程向编辑框写入字符时，另一个线程处于等待状态，如图 6.41 所示。

## 技术要点

本实例主要通过 MFC 的互斥对象 `CMutex` 来实现，该对象实现线程同步主要使用 `Lock` 方法来对资源进行锁定，使用 `Unlock` 来进行解锁。

## 实现过程

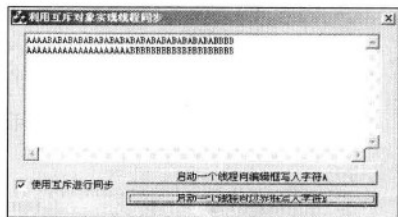


图 6.41 利用互斥对象实现线程同步

(1) 新建一个名为 `MutexSynch` 的对话框 MFC 工程。

(2) 在对话框上添加文本编辑控件，设置 ID 属性为 `IDC_RESULT`，添加成员变量 `m_result`；添加一个复选框控件，设置 ID 属性为 `IDC_MUTEX`，添加成员变量 `m_mutex`；添加两个按钮控件，设置 ID 属性分别为 `IDC_BTSTARTONE` 和 `IDC_BTSTARTTWO`，设置 Caption 属性分别为“启动一个线程向编辑框写入字符 A”和“启动一个线程向边界框写入字符 B”。

(3) `StdAfx.h` 中添加下面语句：

```
#include <afxcmn.h>
```

(4) 在 `MutexSynchDlg.cpp` 文件中添加全局变量声明：

```
CMutex cMutex(FALSE, NULL);
```

(5) 全局线程函数，该函数用于向编辑框中写入字符 A，代码如下：

```
static UINT thread1(LPVOID pParam)
{
 CMutexSynchDlg *pDlg=(CMutexSynchDlg*)pParam; //向线程中传入参数
 CString str;
 if(pDlg->m_mutex==TRUE)
 cMutex.Lock();//加锁
 for(int i=0;i<20;i++)
 {
 pDlg->m_result.GetWindowText(str);
 str+="A";
 pDlg->m_result.SetWindowText(str);
 Sleep(200);
 }
 cMutex.Unlock();//解锁
 return 0;
}
```

(6) 全局线程函数，该函数用于向编辑框中写入字符 B，代码如下：

```
static UINT thread2(LPVOID pParam)
{
 CMutexSynchDlg *pDlg=(CMutexSynchDlg*)pParam; //向线程中传入参数
 CString str;
 if(pDlg->m_mutex==TRUE)
 cMutex.Lock();//加锁
 for(int i=0;i<20;i++)
 {
 pDlg->m_result.GetWindowText(str);
 str+="B";
 pDlg->m_result.SetWindowText(str);
 Sleep(200);
 }
 cMutex.Unlock();//解锁
 return 0;
}
```

(7) “启动一个线程向编辑框写入字符 A”按钮的实现函数,该函数用于启动线程完成相应功能,代码如下:

```
void CMutexSynchDlg::OnStartOne()
{
 AfxBeginThread(thread1,this);//启动一个线程
}
```

(8) “启动一个线程向边界框写入字符 B”按钮的实现函数,该函数用于启动线程完成相应功能,代码如下:

```
void CMutexSynchDlg::OnStartTwo()
{
 AfxBeginThread(thread2,this);//启动一个线程
}
```

(9) 单击复选框控件的实现函数,该函数用于设置是否同步的变量,代码如下:

```
void CMutexSynchDlg::OnMutex()
{
 bmutex=!bmutex;
}
```

### 举一反三

根据本实例,读者可以:

- 利用互斥对象实现进程间共享内存的同步。

## 实例 242 利用临界区实现线程同步

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\06\242

### 实例说明

本实例主要实现如何通过临界区实现线程同步。运行程序,如果使用临界区控制线程同步,则选择“使用临界区”复选框,然后单击相关按钮启动相应的线程向编辑框写入字符,如图 6.42 所示。

### 技术要点

临界区实现线程同步主要通过三个函数,其中 InitializeCriticalSection 函数用于创建临界区,EnterCriticalSection 函数用于进入临界区,LeaveCriticalSection 函数用于退出临界区。当线程进入到临界区后就实现对资源的保护,其他线程就不能对该资源进行读写。

### 实现过程

- (1) 新建一个名为 CriticalSectionSynch 的对话框 MFC 工程。
- (2) 在对话框上添加一个文本编辑控件,设置 ID 属性为 IDC\_RESULT,添加成员变量 m\_result;添加复选框控件,设置 ID 属性为 IDC\_CRITICL,添加成员变量 m\_criticl;添加两个按钮控件,设置 ID 属性分别为 IDC\_BTSTARTONE 和 IDC\_BTSTARTTWO,设置 Caption 属性分别为“启动一个线程向编辑框写入字符 A”和“启动一个线程向编辑框写入字符 B”。

(3) 在 CriticalSectionSynchDlg.cpp 中加入如下代码:

```
#include "afxmt.h"
CRITICAL_SECTION hCritical;
```

(4) 全局线程函数,该函数用于向编辑框中写入字符 A,代码如下:

```
static UINT thread1(LPVOID pParam)
{
 CCriticalSectionSynchDlg *pDlg=(CCriticalSectionSynchDlg*)pParam;
 CString str;
 char buf[MAX_PATH];
 if(pDlg->bcritical==TRUE)
```

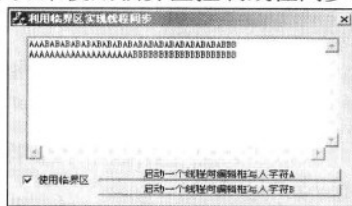


图 6.42 利用临界区实现线程同步



```
EnterCriticalSection(&hCritical); //进入临界区

for(int i=0; i<20; i++)
{
 ::SendMessage(pdlg->m_result.GetSafeHwnd(),
 WM_GETTEXT, sizeof(buf)/sizeof(char), (LPARAM)buf);
 strcat(buf, "A");
 UINT len=strlen(buf);
 buf[len]='\0';
 ::SendMessage(pdlg->m_result.GetSafeHwnd(), WM_SETTEXT, 0, (LPARAM)buf);
 Sleep(200);
}
LeaveCriticalSection(&hCritical); //离开临界区
return 0;
}
```

(5) 全局线程函数, 该函数用于向编辑框中写入字符 B, 代码如下:

```
static UINT thread2(LPVOID pParam)
{
 CCriticalSectionSynchDlg *pdlg=(CCriticalSectionSynchDlg*)pParam;
 CString str;
 char buf[MAX_PATH];
 if(pdlg->bcritical==TRUE)
 EnterCriticalSection(&hCritical); //进入临界区

 for(int i=0; i<20; i++)
 {
 ::SendMessage(pdlg->m_result.GetSafeHwnd(),
 WM_GETTEXT, sizeof(buf)/sizeof(char), (LPARAM)buf);
 strcat(buf, "B");
 UINT len=strlen(buf);
 buf[len]='\0';
 ::SendMessage(pdlg->m_result.GetSafeHwnd(), WM_SETTEXT, 0, (LPARAM)buf);
 Sleep(200);
 }
 LeaveCriticalSection(&hCritical); //离开临界区
 return 0;
}
```

(6) “启动一个线程向编辑框写入字符 A”按钮的实现函数, 该函数用于启动线程 thread1, 代码如下:

```
void CCriticalSectionSynchDlg::OnStartOne()
{
 AfxBeginThread(thread1, this); //开始一个线程
}
```

(7) “启动一个线程向编辑框写入字符 B”按钮的实现函数, 该函数用于启动线程 thread2, 代码如下:

```
void CCriticalSectionSynchDlg::OnStartTwo()
{
 AfxBeginThread(thread2, this);
}
```

(8) 单击复选框控件的实现函数, 该函数用于设置是否同步的变量, 代码如下:

```
void CCriticalSectionSynchDlg::OnCritical()
{
 bcritical=!bcritical;
 InitializeCriticalSection(&hCritical); //初始化临界区
}
```

### 举一反三

根据本实例, 读者可以:

- 利用临界区实现进程间共享内存的同步。

## 实例 243 用信号量实现线程同步

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\06\243

### 实例说明

本实例主要实现文件的复制, 复制的过程中启动了两个线程, 一个线程负责文件的读取,

一个线程负责文件的写入。两个线程通过信号量实现同步，当读取线程运行时通过信号量控制写入线程处于等待状态；当写入线程运行时通过信号量控制读取线程处于等待状态。程序运行如图 6.43 所示。

## 技术要点

本实例通过 `CreateSemaphore` 函数创建信号量句柄来实现。信号量有两种实现方式，一种是本例所使用的通过函数来创建，一种是使用 MFC 对象 `CSemaphore`。`Csemaphore` 对象的使用比较简单，不需要函数创建，要对共享资源进行锁定时使用 `Lock` 方法，解除共享资源的锁定时使用 `UnLock` 方法，在共享资源锁定过程中其他线程不能对资源进行读写。本实例使用的信号量句柄如果要对共享资源进行限制，需要使用 `WaitForSingleObject` 函数，该函数可以控制使用相同信号量句柄的线程处于等待状态，其语法如下：

`DWORD WaitForSingleObject(HANDLE hHandle,DWORD dwMilliseconds);`

参数说明：

- `hHandle`：用于线程同步的内核对象句柄。
- `dwMilliseconds`：指定等待的时间，可以取值 `INFINITE`，实现线程永远等待。

实例中使用 `ReleaseSemaphore` 函数恢复信号量，语法如下：

`BOOL ReleaseSemaphore(HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount);`

参数说明：

- `hSemaphore`：信号量句柄。
- `lReleaseCount`：计数递增数量。
- `lpPreviousCount`：先前计数。

## 实现过程

- (1) 新建一个名为 `SemaphoreSynch` 的对话框 MFC 工程。
- (2) 在对话框上添加 2 个文本编辑控件，设置 ID 属性分别为 `IDC_EDSOURCE` 和 `IDC_EDDES`；添加 3 个按钮控件，设置 ID 属性分别为 `IDC_BTSOURCE`、`IDC_BTDES` 和 `IDC_BTCOPY`；添加 1 个进度条控件，设置 ID 属性为 `IDC_POS`，添加成员变量 `m_pos`。

- (3) 在 `SemaphoreSynchDlg.cpp` 文件中加入如下代码：

```
#include "afxmt.h"
HANDLE hSema; //信号量句柄
```

- (4) 在 `OnInitDialog` 函数中完成信号量的创建，代码如下：

```
BOOL CSemaphoreSynchDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 hSema=CreateSemaphore(NULL,1,1,NULL);//创建信号量
 m_pos.SetRange(0,100);
 m_pos.SetPos(0);
 poslen=0;
 return TRUE;
}
```

- (5) 全局线程函数，该函数用于读取文件操作，代码如下：

```
static UINT thread1(LPVOID pParam)
{
 WaitForSingleObject(hSema,INFINITE);//线程等待
 CSemaphoreSynchDlg *pdlg=(CSemaphoreSynchDlg*)pParam;
 CString tmp;tmp=pdlg->sourcename;
 CFile* readfile;
 readfile=new CFile(tmp,CFile::modeRead);
 readfile->Seek(pdlg->poslen,CFile::begin);
 pdlg->readlen=readfile->Read(pdlg->pvData,512);
 pdlg->poslen+=pdlg->readlen;
 readfile->Close();
 delete readfile;
 ReleaseSemaphore(hSema,1,NULL);//释放信号量
}
```

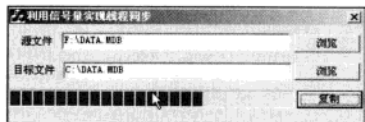


图 6.43 用信号量实现线程同步

```
 return 0;
}
(6) 全局线程函数, 该函数用于写入文件, 代码如下:
static UINT thread2(LPVOID pParam)
```

```
{
 WaitForSingleObject(hSema,INFINITE); //多线程等待
 CSemaphoreSynchDlg *pdlg=(CSemaphoreSynchDlg*)pParam;
 CFile* writefile;
 writefile=new CFile(pdlg->desname,CFile::modeWrite);
 writefile->SeekToEnd();
 writefile->Write(pdlg->pvData,pdlg->readlen);
 writefile->Close();
 delete writefile;
 ReleaseSemaphore(hSema,1,NULL); //释放信号量
 return 0;
}
```

(7) “复制”按钮的实现函数, 该函数用于启动两个线程, 用于读文件和写文件, 代码如下:  
void CSemaphoreSynchDlg::OnCopy()

```
{
 HANDLE hfile=::CreateFile(desname,GENERIC_WRITE|GENERIC_READ,0,0,
 CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,0);
 CloseHandle(hfile);
 CFileStatus status;
 CFile::GetStatus(sourcename,status);
 filelen=status.m_size;
 //计算文件的百分之一大小
 ldiv_t r;
 r=ldiv(filelen,100);
 long pos=r.quot;
 //保存递增的百分比大小
 long ipos;
 ipos=pos;
 int i=0;
 hGlobal = GlobalAlloc(GMEM_MOVEABLE,512); //创建缓存区
 pvData = GlobalLock(hGlobal);
 while(1)
 {
 AfxBeginThread(thread1,this); //开始线程
 AfxBeginThread(thread2,this);
 if(poslen>ipos)
 {
 ipos+=pos;
 i++;
 }
 m_pos.SetPos(i);
 if(poslen==filelen)
 break;
 }
 GlobalUnlock(hGlobal);
 AfxMessageBox("复制完成");
}
```

## 举一反三

根据本实例, 读者可以:

- 用信号量实现进程间共享内存的同步。

## 实例 244 多线程实例

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\06\244

### 实例说明

多线程技术是应用程序开发过程中非常重要的技术。本实例主要实现不同线程独立完成自己的任务, 运行程序, 通过程序中的不同按钮可以启动不同的线程, 每个线程完成任务是向文件中写入指定的字符, 完成的情况都通过进度条显示出来, 如图 6.44 所示。

### 技术要点

本实例主要通过 CreateThread 函数来启动线程, 其语法如下:

```
HANDLE
CreateThread(LPSECURITY_ATTRIBUTES lpThreadAttributes,
 DWORD dwStackSize,LPTHREAD_START_ROUTINE lpStartAddress,
 LPVOID lpParameter,DWORD dwCreationFlags,LPDWORD lpThreadId);
```

参数说明:

- lpThreadAttributes: 指向 SECURITY\_ATTRIBUTES 结构的指针。
- dwStackSize: 指定堆栈的大小。
- lpStartAddress: 指定线程函数。
- lpParameter: 指定线程参数。
- dwCreationFlags: 指定线程的操作。
- lpThreadId: 进程标识。

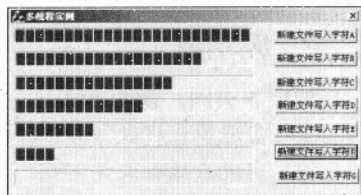


图 6.44 多线程实例

在实例中通过 CreateThread 函数创建线程时将线程的操作指定为 CREATE\_SUSPENDED, 所以还需要使用 ResumeThread 函数使线程处于工作状态。在使线程工作以前可以通过 SetThreadPriority 函数来设置线程的优先级, Windows 系统为线程提供了 7 个级别, 在程序中每个按钮启动的线程都对应 1 个级别。

## 实现过程

- (1) 新建一个名为 ThreadSynch 的对话框 MFC 工程。
- (2) 在对话框上添加 7 个进度条控件, 并添加相应的成员变量; 添加 7 个按钮控件。
- (3) 在头文件 ThreadSynchDlg.h 加入如下变量声明:

```
CString path;
CString str;
```

- (4) 全局线程函数, 该函数用于向文件中写入字符的操作, 同时控制进度条的位置, 代码如下:

```
DWORD WINAPI thread(LPVOID pParam)
{
 CThreadSynchDlg* pDlg=(CThreadSynchDlg*)pParam;
 CString strtmp=pDlg->str;
 CString path=pDlg->path;
 FILE* pf;
 pf=fopen(path.GetBuffer(0),"a");
 CProgressCtrl* p;
 if(strtmp=="A")p=(CProgressCtrl*)&pDlg->m_posa;
 if(strtmp=="B")p=(CProgressCtrl*)&pDlg->m_posb;
 if(strtmp=="C")p=(CProgressCtrl*)&pDlg->m_posc;
 if(strtmp=="D")p=(CProgressCtrl*)&pDlg->m_posd;
 if(strtmp=="E")p=(CProgressCtrl*)&pDlg->m_posf;
 if(strtmp=="F")p=(CProgressCtrl*)&pDlg->m_posf;
 if(strtmp=="G")p=(CProgressCtrl*)&pDlg->m_posg;
 for(int i=0;i<100;i++)
 {
 fprintf(pf,"%s",strtmp);
 p->SetPos(i);
 Sleep(20);
 }
 fclose(pf);
 return 0;
}
```

- (5) “新建文件写入字符 A”按钮的实现函数, 该函数用于启动一个线程向新创建的文本文件中写入字符 A, 代码如下:

```
void CThreadSynchDlg::OnWriteA()
{
 char buf[256];
 ::GetCurrentDirectory(256,buf);//获取当前目录
 strcat(buf,"\\a.txt");
 //创建文件
 HANDLE hfile=::CreateFile(buf,GENERIC_WRITE|GENERIC_WRITE,0,0,
 CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,0);
 CloseHandle(hfile);
 str="A";
 path.Format("%s",buf);
 DWORD nThreadId;
 HANDLE handle=CreateThread(NULL,0,thread,
```



```
(LPVOID)this,CREATE_SUSPENDED,&nThreadId);//创建线程
//设置优先级为高于正常
SetThreadPriority(handle,THREAD_PRIORITY_ABOVE_NORMAL);
ResumeThread(handle);//执行线程
}
```

### 举一反三

根据本实例,读者可以:

- 实现多线程数据的传输;
- 实现多线程文件下载。

## 6.8 鼠标、键盘相关

鼠标键盘是人机交互最直接的工具,本节将通过几个实例介绍与鼠标键盘相关方面的应用知识。

### 实例 245 动画鼠标

本实例是一个非常实用的程序

实例位置:光盘\mingrisoft\06\245

### 实例说明

读者是否对 Windows 98 操作系统中的桌面主题有较深的印象? 各种各样的主题都有着各自特色的桌面、屏幕保护、提示音和漂亮的鼠标,最引人注意的就是其鼠标可以显示不同的动作,并且在没有操作时都一直在动。这样的界面一定会使用户感到新颖的。本实例完成显示动画效果的鼠标。

### 技术要点

静态鼠标文件的扩展名是 ico, 动画鼠标的文件的扩展名是 ani, 本实例主要通过将动画鼠标文件加装到工程中, 然后通过 GetCursor、CopyCursor、LoadCursorFromFile、SetSystemCursor 和 ShowCursor 等函数将动画鼠标显示出来。GetCursor 函数是获得当前鼠标句柄, 通过 CopyCursor 函数可以将鼠标文件暂时保存起来, 以便可以恢复鼠标状态。LoadCursorFromFile 函数将鼠标文件加载进来, 该函数的返回值是鼠标句柄, 通过 SetSystemCursor 函数调用该句柄, 就可以将动画鼠标设置成当前的鼠标状态, 通过 ShowCursor 函数就可以显示出来。

### 实现过程

(1) 新建一个名为 MovieMouse 的对话框 MFC 工程。

(2) 在对话框上添加两个按钮控件, 设置 ID 属性分别为 IDC\_BTMOUSE 和 IDC\_BTRE, 设置 Caption 属性分别为“设置动态光标”和“恢复”。

(3) 在头文件 MovieMouseDlg.h 加入如下变量声明:

```
HCURSOR m_mycursor;
HCURSOR m_oldcursor;
```

(4) 在 OnInitDialog 函数中实现鼠标文件的装载, 代码如下:

```
BOOL CMovieMouseDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_mycursor=::LoadCursorFromFile("mouse.ani");//载入鼠标文件
 if(m_mycursor==NULL)
 AfxMessageBox("装入鼠标文件失败");
 return TRUE;
}
```

(5) 设置动画鼠标, 代码如下:

```
void CMovieMouseDlg::OnBtmouse()
{
```

```

::ShowCursor(false);
m_mycursor=::LoadCursorFromFile("mouse.ani");//载入鼠标文件
HCURSOR cursor=::GetCursor();
m_oldcursor=CopyCursor(cursor);//获取旧鼠标
::SetSystemCursor(m_mycursor,32512);//设置新鼠标
::ShowCursor(true);//显示鼠标
}

```

(6) 恢复鼠标的状态,代码如下:

```

void CMovieMouseDlg::OnBtre()
{
 ::ShowCursor(false);
 ::SetSystemCursor(m_oldcursor,32512);//恢复旧鼠标
 ::ShowCursor(true);
}

```

## 举一反三

根据本实例,读者可以:

- 在程序启动时实现动画鼠标;
- 将动画鼠标应用于任何控件。

## 实例 246 限制鼠标移动区域

本实例是一个非常实用的程序

实例位置: 光盘\mingrisoft\06\246

## 实例说明

鼠标是一个用于操作计算机的外部输入设备。在屏幕中,用一个具有特殊形状的图标来表示当前鼠标。鼠标的移动范围是整个屏幕,用它可以向任意窗口发送指令。一些恶意软件故意改变鼠标移动的区域,将它固定在某个范围内。其实这项技术可以应用在普通的应用软件中。本实例实现将鼠标限制在对话框客户区中。运行程序,如图 6.45 所示。

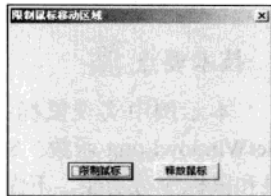


图 6.45 限制鼠标移动区域

## 技术要点

本实例通过 ClipCursor 函数来实现,该函数是将鼠标限制在一定的矩形区域内,只需要调用一个 CRect 对象作为参数即可。

## 实现过程

- (1) 新建一个名为 ClipMouse 的对话框 MFC 工程。
- (2) 在对话框上添加两个按钮控件,设置 ID 属性分别为 IDC\_BTCLIP 和 IDC\_BTRELEASE,设置 Caption 属性分别为“限制鼠标”和“释放鼠标”。

(3) “限制鼠标”按钮的实现函数,该函数将限制鼠标只能在窗体客户区内活动,代码如下:

```

void CClipMouseDlg::OnClip()
{
 CRect rc;
 this->GetWindowRect(rc); //获取窗口大小
 ClipCursor(&rc); //设置鼠标活动区域
}

```

(4) “释放鼠标”按钮的实现函数,该函数将消除对鼠标的限制,代码如下:

```

void CClipMouseDlg::OnRelease()
{
 ClipCursor(NULL);
}

```

## 举一反三

根据本实例,读者可以:

- 锁定计算机。

## 实例 247 鼠标穿透窗体

本实例可以提高基础技能

实例位置: 光盘\mingrisoft\06\247

## 实例说明

用户在使用类似日历的软件时,经常会发现软件的设置功能中包括鼠标穿透功能,该功能的作用是鼠标可以穿透当前窗体,对覆盖下的窗体进行操作,使用户既可以看到日历,又不会影响正常的工作,本例就是使用 Visual C++ 来实现鼠标穿透窗体功能。运行本实例,在窗体上单击鼠标右键,会发现鼠标穿透窗体。实例运行结果如图 6.46 所示。

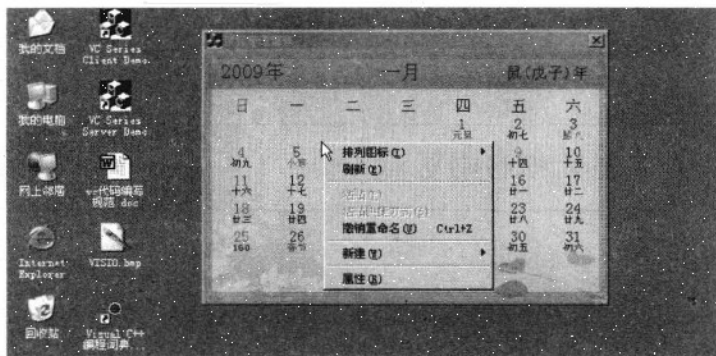


图 6.46 鼠标穿透窗体

## 技术要点

本示例中实现鼠标穿透窗体的功能时,主要用到了 SetLayeredWindowAttributes 函数、GetWindowLong 函数、SetWindowLong 函数和 SetWindowPos 函数,用于设置窗体的半透明效果和鼠标穿透功能,下面对本实例中用到的关键技术进行详细讲解。

## (1) SetLayeredWindowAttributes 函数

SetLayeredWindowAttributes 函数可以设置窗体的半透明效果。该函数并没有被直接封装,需要用户手动从 User32 动态库中导入。首先需要定义一个与 SetLayeredWindowAttributes 函数具有相同函数原型的函数指针,代码如下:

```
typedef BOOL (WINAPI *FSetLayeredWindowAttributes)(HWND,COLORREF,BYTE,DWORD);
FSetLayeredWindowAttributes SetLayeredWindowAttributes;
```

然后调用 LoadLibrary 函数加载 User32 动态库,最后调用 GetProcAddress 函数将 SetLayeredWindowAttributes 指向 User32 动态库中的 SetLayeredWindowAttributes 函数。

## (2) GetWindowLong 函数

GetWindowLong 函数用于获得有关指定窗口的信息。其语法格式如下:

```
LONG GetWindowLong(HWND hWnd, int nIndex);
```

参数说明:

- hWnd: 窗口句柄。
- nIndex: 指定要获得的窗口信息,可选值如表 6.4 所示。

表 6.4

nIndex 参数值表

| 参 数 值       | 描 述      |
|-------------|----------|
| GWL_EXSTYLE | 获得扩展窗口风格 |
| GWL_STYLE   | 获得窗口风格   |

续表

| 参 数 值           | 描 述                          |
|-----------------|------------------------------|
| GWL_WNDPROC     | 获得窗口过程的地址, 或代表窗口过程的地址的句柄     |
| GWL_HINSTANCE   | 获得应用事例的句柄                    |
| GWL_HWNDPAEAENT | 如果父窗口存在, 获得父窗口句柄             |
| GWL_ID          | 获得窗口标识                       |
| GWL_USERDATA    | 获得与窗口有关的 32 位值               |
| DWL_DLGPROC     | 获得对话框过程的地址, 或一个代表对话框过程的地址的句柄 |
| DWL_MSGRESULT   | 获得在对话框过程中一个消息处理的返回值          |
| DWL_USER        | 获得应用程序私有的额外信息                |

### (3) SetWindowLong 函数

SetWindowLong 函数用于设置窗口风格。其语法格式如下:

```
LONG SetWindowLong(HWND hWnd, int nIndex, LONG dwNewLong);
```

参数说明:

- hWnd: 窗口句柄。
- nIndex: 要设置的信息。
- dwNewLong: 指定的窗口风格。

### (4) SetWindowPos 函数

SetWindowPos 函数用于设置窗口的显示位置。其语法格式如下:

```
BOOL SetWindowPos(HWND hWnd, HWND hWndInsertAfter, int X, int Y, int cx, int cy, UINT uFlags);
```

参数说明:

- hWnd: 窗口句柄。
- hWndInsertAfter: 在窗口次序中的位于被置位的窗口前的窗口句柄。可选值如表 6.5 所示。

表 6.5 hWndInsertAfter 参数值表

| 参数值            | 描述                                |
|----------------|-----------------------------------|
| HWND_BOTTOM    | 将窗口置于窗口次序的底部                      |
| HWND_NOTOPMOST | 将窗口置于所有非顶层窗口之上 (即在所有顶层窗口之后)       |
| HWND_TOP       | 将窗口置于窗口次序的顶部                      |
| HWND_TOPMOST   | 将窗口置于所有非顶层窗口之上。即使窗口未被激活窗口也将保持顶级位置 |

- X: 以客户坐标指定窗口新位置的左边界。
- Y: 以客户坐标指定窗口新位置的顶边界。
- cx: 以像素指定窗口的新的宽度。
- cy: 以像素指定窗口的新的高度。
- uFlags: 窗口尺寸和定位的标志。

## 实现过程

(1) 新建一个基于对话框的应用程序, 将其窗体标题改为“鼠标穿透窗体”。

(2) 向工程中导入一个 BMP 位图资源, 并向对话框中添加一个图片控件, 设置其 Type 属性为 Bitmap, 设置其 Image 属性为 IDB\_BITMAP1。

(3) 主要程序代码。

在主窗体初始化时, 设置窗体半透明, 并设置窗体具有鼠标穿透功能, 其实现代码如下:

```
//设置窗口扩展风格
```

```
SetWindowLong(GetSafeHwnd(),GWL_EXSTYLE,
```

```
GetWindowLong(GetSafeHwnd(),GWL_EXSTYLE)| 0x80000);
```

```
//设置扩展风格0x80000
```



```

typedef BOOL (WINAPI *FSetLayeredWindowAttributes)(HWND,COLORREF, BYTE,DWORD);
FSetLayeredWindowAttributes SetLayeredWindowAttributes; //声明函数
HINSTANCE hInst = LoadLibrary("User32.DLL"); //加载动态链接库
SetLayeredWindowAttributes = (FSetLayeredWindowAttributes)
GetProcAddress(hInst,"SetLayeredWindowAttributes"); //获得函数地址
if(SetLayeredWindowAttributes)
 SetLayeredWindowAttributes(GetSafeHwnd(),RGB(0,0,0),160,2); //设置窗体半透明
FreeLibrary(hInst);
long dwNewLong;
dwNewLong = GetWindowLong(this->m_hWnd, GWL_EXSTYLE); //获得当前窗体风格
dwNewLong |= WS_EX_TRANSPARENT; //设置鼠标穿透
SetWindowLong(this->m_hWnd, GWL_EXSTYLE, dwNewLong); //设置窗体风格
CRect rect;
GetWindowRect(&rect);
::SetWindowPos(AfxGetMainWnd()->m_hWnd,HWND_TOPMOST,rect.left,rect.top,rect.Width()
,rect.Height(),SWP_SHOWWINDOW); //将窗体置顶

```

### 举一反三

根据本实例，读者可以：

- 创建动感界面。

## 实例 248 设置鼠标形状

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\06\248

### 实例说明

鼠标是一个用于操作计算机的外部输入设备。在屏幕中，用一个具有特殊形状的图标来表示当前鼠标。并且当鼠标在不同的窗体控件上移动时可以显示不同形状的鼠标。其实这项技术可以应用在普通的应用软件中。运行程序，如图 6.47 所示。

### 技术要点

在程序运行时通过动态载入鼠标的资源，再通过窗体的 OnSetCursor 消息事件中使用 SetCursor 函数实现鼠标形状的动态改变。



图 6.47 设置鼠标形状

### 实现过程

- (1) 新建一个基于对话框 MFC 工程。
- (2) 在对话框上添加 3 个按钮控件，设置 Caption 属性分别为“改变鼠标”、“恢复鼠标”和“关闭”。

- (3) 当需要改变鼠标形状时载入鼠标资源，并设默认鼠标标记为 false，代码如下：

```

void CSetCursorDlg::OnChangeCursor()
{
 m_hCursor = ::AfxGetApp()->LoadCursor(IDC_CURSOR1);
 isDefault = false;
}

```

- (4) 在窗体的 OnSetCursor 消息事件中改变当前鼠标的形状，代码如下：

```

BOOL CSetCursorDlg::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
 if(!isDefault)
 {
 ::SetCursor(m_hCursor);
 return true;
 }
 else
 return CDialog::OnSetCursor(pWnd, nHitTest, message);
}

```

### 举一反三

根据本实例，读者可以：

● 锁定鼠标的活动范围。

## 实例 249 控制键盘指示灯

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\06\249

### 实例说明

在控制键盘指示灯的过程中必须由使用人员手动的按下键盘上相应的按键才可以，这样就必须使鼠标操作和键盘操作来回切换。本实例通过程序直接控制键盘指示灯的状态。运行程序，如图 6.48 所示。

### 技术要点

通过 API 函数 GetKeyboardState 可以获取当前键盘按键的状态，如果键盘上 SCROLL、CAPITAL 和 NUMLOCK 3 个指示灯亮着，表示 3 个指示灯所对应的键盘按键正处在按下的状态。如果想要控制指示灯的亮与灭，只要使用 keybd\_event 函数来模拟键盘按下的动作。

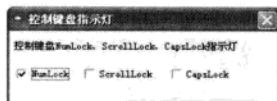


图 6.48 控制键盘指示灯

### 实现过程

- (1) 新建一个基于对话框 MFC 工程。
- (2) 在对话框上添加 1 个静态文本框控件，设置其 Caption 属性为“控制键盘 NumLock、ScrollLock、CapsLock 指示灯”；添加 3 个复选框控件。

(3) 主要程序代码。

```
void CKeyboardNumLampDlg::OnNumLock()
{
 keybd_event(VK_NUMLOCK, 0x45, KEYEVENTF_EXTENDEDKEY | 0, 0);
 keybd_event(VK_NUMLOCK, 0x45, KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, 0);
}

void CKeyboardNumLampDlg::OnCapLock()
{
 keybd_event(VK_CAPITAL, 0x45, KEYEVENTF_EXTENDEDKEY | 0, 0);
 keybd_event(VK_CAPITAL, 0x45, KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, 0);
}

void CKeyboardNumLampDlg::OnScrollLock()
{
 keybd_event(VK_SCROLL, 0x45, KEYEVENTF_EXTENDEDKEY | 0, 0);
 keybd_event(VK_SCROLL, 0x45, KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, 0);
}

BOOL CKeyboardNumLampDlg::PreTranslateMessage(MSG* pMsg)
{
 if(pMsg->message == WM_KEYUP) // 键盘按下
 {
 BYTE keyState[256];
 GetKeyboardState((LPBYTE)&keyState);
 if(keyState[VK_NUMLOCK] & 1)
 m_num.SetCheck(true);

 else
 m_num.SetCheck(false);
 if(keyState[VK_SCROLL] & 1)
 m_scroll.SetCheck(true);
 else
 m_scroll.SetCheck(false);
 if(keyState[VK_CAPITAL] & 1)
 m_cap.SetCheck(true);
 else
 m_cap.SetCheck(false);
 }
 return CDialog::PreTranslateMessage(pMsg);
}
```

## 举一反三

根据本实例,读者可以:

- 制作虚拟键盘程序。

## 6.9 动态链接库

动态库在程序设计中起到了非常重要的作用,由于动态库是可以进程内共享所以为模块化编程带来了方便。

## 实例 250 访问 DLL 中的位图

本实例是一个非常实用的程序

实例位置: 光盘\mingrisoft\06\250

## 实例说明

动态库与应用程序一样,可以存储各种资源,包括位图资源。那么如何从动态库中获取位图资源呢?本例实现了该功能,程序运行效果如图 6.49 所示。

## 技术要点

不管是在应用程序工程中还是在动态库工程中资源的读取都是通过 LoadBitmap 函数来获以的。该函数接收两个参数:一个是工程句柄;另一个是资源 ID。所以在读取动态库工程中的资源信息时需要先将动态库载入,再通过 LoadBitmap 函数来获取动态库中的资源。



图 6.49—访问 DLL 中的位图

## 实现过程

- (1) 新建一个名为 MFCDLL 的 MFC 动态库工程。
- (2) 在动态库中载入位图资源。
- (3) 新建一个名为 FetchBmp 的 MFC 对话框工程。
- (4) 在对话框中添加两个按钮,设置其 Caption 属性为“显示”和“取消”。
- (5) 载入动态库获取资源句柄,实现代码如下:

```
HINSTANCE hInstance = LoadLibrary("MFCDLL.dll");
m_hBitmap = LoadBitmap(hInstance, MAKEINTRESOURCE(1000));
FreeLibrary(hInstance);
```

- (6) 单击“显示”按钮通过资源句柄将位图绘制在窗体上,实现代码如下:

```
void CFetchBmpDlg::OnOK()
{
 if(m_hBitmap)
 {
 CBitmap bmp;
 bmp.Attach(m_hBitmap);
 CDC* pDC = GetDC();
 CDC memDC;
 memDC.CreateCompatibleDC(pDC);
 memDC.SelectObject(&bmp);
 pDC->BitBlt(0,0,500,600,&memDC,0,0,SRCCOPY);
 bmp.Detach();
 memDC.DeleteDC();
 }
}
```

## 举一反三

根据本实例,读者可以:

- 通过动态库实现多语言程序;

- 在动态库中存储窗体皮肤资源。

## 实例 251 从 DLL 中导出类对象

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\06\251

### 实例说明

创建可以导出 Visual C++ 类的动态链接库可以使用 Win32 Dynamic-Link Library 工程向导，使用 Win32 Dynamic-Link Library 工程向导创建链接库非常简单，只需要一步，工程向导对话框如图 6.50 所示。

### 技术要点

一个能够从动态链接库中导出的 Visual C++ 类，关键之处是使用了语句 `__declspec(dllexport)`，例如实例中创建了一个 `CMath` 类，并在 `CMath` 类中定义了两个成员函数，一个是 `GetMaxCommDiv` 用来计算最大公约数，一个 `GetMinCommMul` 用来计算最小公倍数。如果想导出 `CMath` 类，就需要在 `CMath` 的前面，`class` 关键字的后面添加语句

`__declspec(dllexport)`，实例中 `ExportClass.h` 文件中定义了可以导出的 `CMath` 类，具体代码如下：

```
class __declspec(dllexport) CMath
{
public:
 int GetMaxCommDiv(int a,int b); //最大公约数
 int GetMinCommMul(int a,int b); //最小公倍数
};
```

如果只是想导出类中的成员函数，那么就只想在导出的函数名前加 `__declspec(dllexport)` 语句，例如想导出 `CMath` 类中的 `GetMinCommMu` 成员函数，其实现代码如下：

```
class (dllexport) CMath
{
public:
 int GetMaxCommDiv(int a,int b);
 int __declspec GetMinCommMul(int a,int b);
};
```

使用 `__declspec(dllexport)` 语句的 `GetMinCommMul` 成员函数就可以像一个全局函数一样被调用。

### 实现过程

- (1) 使用 Win32 Dynamic-Link Library 工程向导创建一个空的 DLL 工程，工程名为 `ExportClass`。
- (2) 手动添加一个头文件和一个实现文件，实例中添加的实现文件为 `ExportClass.cpp`，添加的头文件为 `ExportClass.h`。

- (3) 定义导出类的类型定义，代码如下：

```
class __declspec(dllexport) CMath
{
public:
 int GetMaxCommDiv(int a,int b); //最大公约数
 int GetMinCommMul(int a,int b); //最小公倍数
};
```

- (4) 实现导出类的实体文件，代码如下：

```
#include "ExportClass.h"
#include "windows.h"//使用了MessageBox函数所以要加入此头文件引用
//最大公约数
int CMath::GetMaxCommDiv(int a,int b)
{
 int x=1; //a, b保存相除后的结果
 int ires;//保存结构
 if(a<b)
 {
 MessageBox(NULL,"a不能小于b","出错",MB_OK);
 return 0;
```

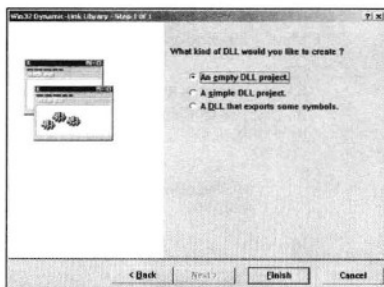


图 6.50 从 DLL 中导出类对象





```

 }
 if(b==0)
 {
 MessageBox(NULL,"b不能为0","出错",MB_OK);
 return 0;
 }
 while(x!=0)
 {
 x=a%b;
 a=b;
 ires=b;
 b=x;
 }
 return ires;
}
//最小公倍数
int CMath::GetMinCommMul(int a,int b)
{
 int x=1; //a, b保存相除后的结果
 int ires;//保存结果
 int m,n;//存储临时的a, b值
 m=a;n=b;
 if(a<b)
 {
 MessageBox(NULL,"a不能小于b","出错",MB_OK);
 return 0;
 }
 if(a==0)
 {
 MessageBox(NULL,"a不能为0","出错",MB_OK);
 return 0;
 }
 if(b==0)
 {
 MessageBox(NULL,"b不能为0","出错",MB_OK);
 return 0;
 }
 while(x!=0)
 {
 x=a%b;
 a=b;
 ires=b;
 b=x;
 }
 ires=(m*n)/(ires);
 return ires;
}

```

(5) 新建一个对话框工程，将 ExportClass.h、ExportClass.DLL、ExportClass.Lib 文件复制到工程目录下。在 StdAfx.h 头文件中加入如下代码：

```

#include "ExportClass.h"
#pragma comment(lib,"ExportClass")

```

(6) 这样就可以在工程中操作从 DLL 中导出的类了，实现代码如下：

```

void CTestDllDlg::OnOK()
{
 CString a,b,c,d,value1,value2;
 m_a.GetWindowText(a);
 m_b.GetWindowText(b);
 m_c.GetWindowText(c);
 m_d.GetWindowText(d);
 if (a != "" && b != "")
 {
 int value = m_math.GetMaxCommDiv(atoi(a),atoi(b));
 value1.Format("%d",value);
 m_value1.SetWindowText(value1);
 }
 if (c != "" && d != "")
 {
 int value = m_math.GetMinCommMul(atoi(c),atoi(d));
 value2.Format("%d",value);
 m_value2.SetWindowText(value2);
 }
}

```

## 举一反三

根据本实例，读者可以：

- 利用 DLL 实现插件式编程。

## 第 7 章 注册表

- 显示与隐藏
- IE 浏览器设置
- 文件控制
- 游戏设置
- 应用软件设置

Visual C++

## 7.1 显示与隐藏

为了不让其他用户使用计算机中的某些功能,可以通过注册表技术将这些功能隐藏。本节实例将介绍如何通过 Visual C++实现隐藏或显示计算机中的某些功能。

### 实例 252

### 隐藏、显示“我的电脑”、“回收站”、“网上邻居”

本实例是一个显示、隐藏计算机中功能的程序

实例位置: 光盘\mingrisoft\07\252

#### 实例说明

在 Windows 系统的桌面上一般会有“我的电脑”、“回收站”和“网上邻居”3个图标,这3个图标用一般删除文件的方法来删除是无法实现的。本实例实现了将这3个图标隐藏起来的功能。运行程序,如图 7.1 所示,单击相应的按钮,重新启动计算机后,即可隐藏或显示“我的电脑”、“回收站”和“网上邻居”。

#### 技术要点

##### (1) 隐藏、显示“我的电脑”。

在 Visual C++中使用 API 函数 RegSetValueEx 在注册表中“HKEY\_CLASSES\_ROOT\CLSID\{20D04FE0-3AEA-1069-A2D8-08002B30309D}\ShellFolder”的子项中新建一个 REG-DWORD 值项 Attributes,将该值设置为“0xFFFFFFFF”即可隐藏“我的电脑”,如果要显示“我的电脑”,就将该值设置为“0”。

##### (2) 隐藏、显示“回收站”。

在 Visual C++中使用 API 函数 RegDeleteKey 删除注册表中“HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\Desktop\NameSpace”次级主键中的{645FF040-5081-101B-9F08-00AA002F954E}项,可以隐藏“回收站”,重新创建该项,可以显示“回收站”。

##### (3) 隐藏、显示“网上邻居”。

在 Visual C++中使用 API 函数 RegSetValueEx 在注册表“HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer”中新建一个 REG-DWORD(双字节)值 NoNetHood,它的默认值为 0,表示正常显示桌面上的“网上邻居”,将该值设置为 1,即可隐藏桌面上的“网上邻居”。

RegCreateKey 函数用于打开或创建注册表项,语法如下:

```
LONG RegCreateKey(HKEY hKey,LPCTSTR lpSubKey,PHKEY phkResult);
```

参数说明:

- hKey: 注册表主键。
- lpSubKey: 指向主键下子项的字符串指针。
- phkResult: 返回打开或新建的 HKEY 对象。

RegSetValueEx 函数用于设置注册表值项的数据,语法如下:

```
LONG RegSetValueEx(HKEY hKey,LPCTSTR lpValueName,
DWORD Reserved,DWORD dwType,CONST BYTE *lpData,DWORD cbData);
```

参数说明:

- hKey: 注册表主键。
- lpValueName: 指向主键下子项的字符串指针。



图 7.1 实例运行结果

- Reserved: 保留。
- dwType: 值项的类型。
- lpData: 指向数据缓存的指针。
- cbData: 数据缓存的大小。

RegDeleteKey 函数用于删除注册表的子项, 语法如下:

```
LONG RegDeleteKey(HKEY hKey, LPCTSTR lpSubKey);
```

参数说明:

- hKey: 注册表主键。
- lpSubKey: 指向主键下子项的字符串指针。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加 6 个按钮, 设置 ID 属性分别为 IDC\_BTHIDECOMPUTER、IDC\_BTSHOWCOMPUTER、IDC\_BTHIDERECY、IDC\_BTSHOWRECY、IDC\_BTHIDENET 和 IDC\_BTSHOWNET, 设置 Caption 属性分别为“隐藏我的电脑”、“显示我的电脑”、“隐藏回收站”、“显示回收站”、“隐藏网上邻居”、“显示网上邻居”。

(3) 主要程序代码如下:

```
void CHideMyComputerDlg::OnHideComputer()
{
 //隐藏我的电脑
 HKEY sub;
 DWORD val=0xffffffff;
 CString skey="CLSID\\{20D04FE0-3AEA-1069-A2D8-08002B30309D}\\ShellFolder"; //设置字符串
 ::RegCreateKey(HKEY_CLASSES_ROOT,skey,&sub); //打开注册表项
 RegSetValueEx(sub,"Attributes",NULL,REG_DWORD,(BYTE*)&val,sizeof(DWORD)); //设置键值
 ::RegCloseKey(sub);
}

void CHideMyComputerDlg::OnShowComputer()
{
 // 显示我的电脑
 HKEY sub;
 DWORD val=0;
 CString skey="CLSID\\{20D04FE0-3AEA-1069-A2D8-08002B30309D}\\ShellFolder"; //设置字符串
 ::RegCreateKey(HKEY_CLASSES_ROOT,skey,&sub); //打开注册表项
 RegSetValueEx(sub,"Attributes",NULL,REG_DWORD,(BYTE*)&val,sizeof(DWORD)); //设置键值
 ::RegCloseKey(sub);
}

void CHideMyComputerDlg::OnHideRecy()
{
 //隐藏回收站
 CString skey="Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Desktop\\NameSpace\\{645FF040-5081-101B-9F08-00AA002F954E}";
 LONG ReturnValue=::RegDeleteKey(HKEY_LOCAL_MACHINE,skey);
 if(ReturnValue!=ERROR_SUCCESS)
 {
 AfxMessageBox("隐藏失败");
 }
}

void CHideMyComputerDlg::OnShowRecy()
{
 // 显示回收站
 HKEY sub;
 CString skey="Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Desktop\\NameSpace\\{645FF040-5081-101B-9F08-00AA002F954E}";
 LONG ReturnValue=::RegCreateKey(HKEY_LOCAL_MACHINE,skey,&sub);
 if(ReturnValue!=ERROR_SUCCESS)
 {
 AfxMessageBox("创建注册表项失败");
 }
 ::RegCloseKey(sub);
}
```



```

}

void CHideMyComputerDlg::OnHideNet()
{
 // 隐藏网上邻居
 HKEY sub;
 DWORD val=1;
 CString key="Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer\\";
 ::RegCreateKey(HKEY_CURRENT_USER,key,&sub);
 RegSetValueEx(sub,"NoNetHood",NULL,REG_DWORD,(BYTE*)&val,sizeof(DWORD));
 ::RegCloseKey(sub);
}

void CHideMyComputerDlg::OnShowNet()
{
 // 显示网上邻居
 HKEY sub;
 DWORD val=0;
 CString key="Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer\\";
 ::RegCreateKey(HKEY_CURRENT_USER,key,&sub);
 RegSetValueEx(sub,"NoNetHood",NULL,REG_DWORD,(BYTE*)&val,sizeof(DWORD));
 ::RegCloseKey(sub);
}

```

### 举一反三

根据本实例，读者可以：

- 开发隐藏“我的文档”图标程序；
- 开发利用注册表记录数据的程序；
- 开发带图标的按钮的程序。

## 实例 253 隐藏、显示驱动器

本实例可以提高计算机的安全性

实例位置：光盘\mingrisoft\07\253

### 实例说明

默认情况下，在“我的电脑”中显示所有驱动器，但通过本实例可以隐藏这些驱动器及其中的所有文件，防止驱动器中的文件被浏览、修改或删除。单击“隐藏所有驱动器”按钮，重新启动计算机后，“我的电脑”中的所有磁盘将不再显示。程序运行结果如图 7.2 所示。

### 技术要点

本实例应用 RegCreateKey 函数和 RegSetValueEx 函数对注册表进行修改。打开注册表 HKey\_current\_user\Software\microsoft\windows\currentVersion\ Policies\ Explorer 子项，设置值项 NoDrives 的数据。

**注意：**在进行此项操作之前，首先对注册表文件进行备份或将本实例复制到桌面上，因为在进行此项操作之后，“我的电脑”中的所有驱动器将被隐藏，用户就无法使用驱动器来打开文件了。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框上添加两个按钮，设置 ID 属性分别为 IDC\_BTHIDE1 和 IDC\_BTSHOW；设置 Caption 属性分别为“隐藏所有驱动器”和“显示所有驱动器”。

(3) 添加“隐藏所有驱动器”按钮的实现函数，该函数隐藏系统中的所有驱动器，代码如下：

```

void CHideDriverDlg::OnHide()
{
 HKEY sub;

```

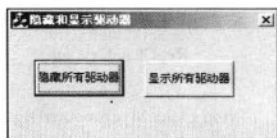


图 7.2 隐藏、显示驱动器





单的隐藏,代码如下:

```
void CIERRightMenuDlg::OnHide()
{
 HKEY sub;
 DWORD cb;
 DWORD val=1;
 SECURITY_ATTRIBUTES sa;
 sa.nLength = sizeof(SECURITY_ATTRIBUTES);
 sa.bInheritHandle = TRUE;
 sa.lpSecurityDescriptor = NULL;
 CString key="Software\\Policies\\Microsoft\\Internet Explorer\\Restrictions";
 ::RegCreateKeyEx(HKEY_CURRENT_USER,skey,0L,
 "",REG_OPTION_NON_VOLATILE,KEY_ALL_ACCESS,&sa,&sub,&cb);
 //设置键值
 RegSetValueEx(sub,"NoBrowserContextMenu",NULL,REG_DWORD,(BYTE*)&val,sizeof(DWORD));
}
```

(5) 添加“显示 IE 右键菜单”按钮的实现函数,该函数向注册表中写入数据完成右键菜单的显示,代码如下:

```
void CIERRightMenuDlg::OnShow()
{
 HKEY sub;
 DWORD cb;
 DWORD val=0;
 SECURITY_ATTRIBUTES sa;
 sa.nLength = sizeof(SECURITY_ATTRIBUTES);
 sa.bInheritHandle = TRUE;
 sa.lpSecurityDescriptor = NULL;
 CString key="Software\\Policies\\Microsoft\\Internet Explorer\\Restrictions";
 ::RegCreateKeyEx(HKEY_CURRENT_USER,skey,0L,
 "",REG_OPTION_NON_VOLATILE,KEY_ALL_ACCESS,&sa,&sub,&cb);
 //设置键值
 RegSetValueEx(sub,"NoBrowserContextMenu",NULL,REG_DWORD,(BYTE*)&val,sizeof(DWORD));
}
```

### 举一反三

根据本实例,读者可以:

- 实现打开或关闭 IE 窗口。

## 实例 256

### 设置 IE 浏览器的默认主页

本实例可以提高计算机的安全性

实例位置: 光盘\mingrisoft\07\256

### 实例说明

本实例实现对 IE 中默认主页的修改。在文本框中输入主页地址,单击“设置主页”按钮,相关信息就会写入注册表,IE 会自动打开设置的主页,运行程序如图 7.5 所示。

### 技术要点

本实例应用 RegCreateKey 函数和 RegSetValueEx 函数对注册表进行修改。打开注册表 HKEY\_CURRENT\_USER\Software\Microsoft\Internet Explorer\Main 子项。设置 Main 子项下的 Start Page 值项的数据。

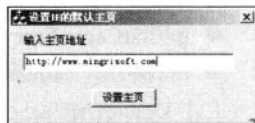


图 7.5 设置 IE 的默认主页

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框上添加静态文本控件,设置 Caption 属性为“输入主页地址”;添加编辑框控件,设置 ID 属性为 IDC\_EDSTARTPAGE;添加按钮控件,设置 ID 属性为 IDC\_BTSET,设置 Caption 属性为“设置主页”。
- (3) 添加按钮的实现函数 OnSetStartPage。



(4) 主要程序代码如下:

```
void CIEStartPageDlg::OnSetStartPage()
{
 CString strstartpage;
 GetDlgItem(IDC_EDSTARTPAGE)->GetWindowText(strstartpage);
 HKEY sub;
 CString skey="Software\\Microsoft\\Internet Explorer\\Main";
 ::RegCreateKey(HKEY_CURRENT_USER,skey,&sub);
 RegSetValueEx(sub,"Start Page",NULL,REG_SZ, (BYTE*)strstartpage.GetBuffer(strstartpage.GetLength()),
 strstartpage.GetLength()); //写入数据
 RegCloseKey(sub);
}
```

## 举一反三

根据本实例,读者可以:

- 设置应用程序默认打开的文件。

## 实例 257 清空上网历史记录

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\07\257

## 实例说明

IE 浏览器的地址栏会记录一些访问过的网页地址,本实例主要通过修改注册表将这些数据清空。运行程序,单击“清空上网历史记录”按钮,程序会删除历史记录数据。程序运行结果如图 7.6 所示。

## 技术要点

本实例应用 RegDeleteKey 函数对注册表进行修改,注册表中 HKEY\_CURRENT\_USER\Software\Microsoft\Internet Explorer\TypedURLs 子项下的值项记录着上网历史记录,将该子项删除,上网历史记录就会被清空。本实例还应用 FindFirstUrlCacheEntry 函数、FindFirstUrlCacheEntryEx 函数、DeleteUrlCacheEntry 函数和 FindNextUrlCacheEntry 函数实现对上网缓存数据的修改。

(1) FindFirstUrlCacheEntry 函数。用来获得或打开上网 URL 缓存句柄,语法如下:

```
HANDLE FindFirstUrlCacheEntry (IN LPCSTR lpszUrlSearchPattern,
 OUT LPINTERNET_CACHE_ENTRY_INFO lpFirstCacheEntryInfo,
 IN OUT LPDWORD lpdwFirstCacheEntryInfoBufferSize);
```

参数说明:

- lpszUrlSearchPattern: 缓存句柄的模式名称,参数值有“cookie:”、“visited:”、“\*.\*”。
- lpFirstCacheEntryInfo: 缓存句柄,如果为 NULL,函数将返回获得的句柄。
- lpdwFirstCacheEntryInfoBufferSize: 返回缓存句柄的大小。

(2) DeleteUrlCacheEntry 函数。用来删除缓存,语法如下:

```
BOOL DeleteUrlCacheEntry(IN LPCSTR lpszUrlName);
```

参数说明:

- lpszUrlName: 即将删除的缓存名称。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框上添加两个按钮控件, ID 分别为 IDC\_BTCCLEAR 和 IDC\_BTEXIT, Caption 属性分别为“清空上网历史记录”和“退出”。
- (3) 通过类向导添加“清空上网历史记录”按钮的实现函数 OnClear; 添加“退出”按钮的实现函数 OnExit。

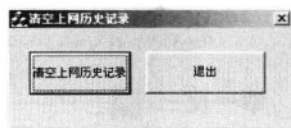


图 7.6 清空上网历史记录

(4) “清空上网历史记录”按钮的实现函数完成历史记录的删除,代码如下:

```
void CDeleteURLDlg::OnClear()
{
 //浏览器地址栏历史记录
 CString skey="Software\\Microsoft\\Internet Explorer\\TypedURLs";
 ::RegDeleteKey(HKEY_CURRENT_USER,skey);
 //清除cookie和临时文件
 HANDLE hEntry;
 LPINTERNET_CACHE_ENTRY_INFO lpCacheEntry=NULL;
 DWORD dwEntrySize;

 dwEntrySize=0;
 hEntry=FindFirstUrlCacheEntry(NULL, NULL, &dwEntrySize); //获得上网URL缓存句柄
 lpCacheEntry=(LPINTERNET_CACHE_ENTRY_INFO)new char[dwEntrySize];
 hEntry=FindFirstUrlCacheEntryEx(NULL,0,NORMAL_CACHE_ENTRY|URLHISTORY_CACHE_ENTRY,0,
 lpCacheEntry,&dwEntrySize,NULL,NULL,NULL);
 do
 {
 DeleteUrlCacheEntry(lpCacheEntry->lpszSourceUrlName); //删除缓存
 dwEntrySize = 0;
 FindNextUrlCacheEntry(hEntry, NULL, &dwEntrySize);
 ZeroMemory(lpCacheEntry,dwEntrySize);
 }
 while(FindNextUrlCacheEntry(hEntry, lpCacheEntry, &dwEntrySize));
 delete lpCacheEntry;
}
```

(5) “退出”按钮的实现函数,代码如下:

```
void CDeleteURLDlg::OnExit()
{
 this->OnCancel();
}
```

### 举一反三

根据本实例,读者可以:

- 清空“我的文档”中的历史记录;
- 清空“开始”菜单运行中的记录。

## 7.3 文件控制

计算机中的数据都是以文件的形式存储在磁盘介质中的,用户的应用程序经常与文件打交道,对文件进行控制就显得比较重要。本节通过两个实例来介绍如何通过注册表对文件进行控制。

### 实例 258 如何建立文件关联

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\07\258

#### 实例说明

所谓文件关联,就是某种类型的文件与某个应用程序关联起来。当在资源管理器中双击这种类型的文件时,由资源管理器启动相关联的应用程序,并打开该文件。例如,在资源管理器中双击扩展名为.jpg的文件,资源管理器将启动相应的看图软件,并在看图软件中打开该文件。运行程序,在文本框中输入相关数据,然后单击“建立关联”按钮,即可建立指定文件的关联。程序运行结果如图 7.7 所示。

#### 技术要点

默认情况下扩展名为.txt的文本文件是用记事本打开的,通过使用本实例可以将扩展名为.txt的文本文件用其他文本编辑软件打开。



首先在注册表 HKEY\_CLASSES\_ROOT 根键下找到.txt 子项, 该子项有“默认”和“Content Type”两个值项, “默认”值项的数据值是 txtfile。

然后在注册表 HKEY\_CLASSES\_ROOT 根键下找到 txtfile 子项, 程序主要修改该子项下的 DefaultIcon 子项和 Shell 子项下的内容, DefaultIcon 子项是设置扩展名为.txt 的文本文件在资源管理器中显示的图标; Shell 子项下还有两个重要的子项, 即 open 子项和 print 子项, 分别用来设置“打开”命令和“打印”命令。

实现将扩展名为.txt 的文本文件用其他文本编辑软件打开, 就是修改 open 子项的内容, 具体操作是打开注册表 HKEY\_CLASSES\_ROOT\txtfile\Shell\open\command 子项, 修改该子项中的“默认”值项的数据值, 设置为文本编辑软件的可执行文件全路径加相应的参数即可。例如, 注册表原来的数据值是 C:\WINDOWS\NOTEPAD.EXE /p %1, 其中“%1”表示文本文件的路径。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加 3 个按钮控件, 设置 ID 属性分别为 IDC\_ADDICON、IDC\_REL 和 IDC\_EXIT, 设置 Caption 属性分别为“...”、“建立关联”和“退出”; 添加 7 个静态文本控件, 设置 Caption 属性分别为“扩展名”、“默认值”、“类型描述”、“Shell”、“打开命令”、“打印命令”和“文件图标”; 添加 7 个编辑框控件, 设置 ID 属性分别为 IDC\_EDNAME、IDC\_EDVALUE、IDC\_EDSCRI、IDC\_EDSHELL、IDC\_EDCOMMAND、IDC\_EDPRINT 和 IDC\_EDICON。

(3) 添加“...”按钮的实现函数 OnAddicon, 添加“建立关联”按钮的实现函数 OnRel, 添加“退出”按钮的实现函数 OnExit。

(4) 在头文件中添加变量声明:

CString strname;

(5) “...”按钮的实现函数实现添加一个图标文件, 代码如下:

```
void CFileRelatDlg::OnAddicon()
{
 CFileDialog file(FALSE, "ico", NULL, OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,
 "ICON 文件 (*.ico)|*.ico|", this);
 if(file.DoModal() == IDOK)
 {
 strname = file.GetPathName();
 GetDlgItem(IDC_EDICON) -> SetWindowText(strname);
 }
}
```

(6) “建立关联”按钮的实现函数向注册表中写入数据, 完成文件的关联, 代码如下:

```
void CFileRelatDlg::OnRel()
{
 // 获得控件中数据
 CString stricon, strname, strvalue, strscri, strshell, strcommand, strprint;
 GetDlgItem(IDC_EDNAME) -> GetWindowText(strname);
 GetDlgItem(IDC_EDVALUE) -> GetWindowText(strvalue);
 GetDlgItem(IDC_EDSCRI) -> GetWindowText(strscri);
 GetDlgItem(IDC_EDSHELL) -> GetWindowText(strshell);
 GetDlgItem(IDC_EDCOMMAND) -> GetWindowText(strcommand);
 GetDlgItem(IDC_EDPRINT) -> GetWindowText(strprint);
 GetDlgItem(IDC_EDICON) -> GetWindowText(stricon);
 DWORD c = 1;
 HKEY sub;
 CString skey = strname;
 // 建立扩展名
 ::RegCreateKey(HKEY_CLASSES_ROOT, skey, &sub);
 // 设置扩展名的默认值
```

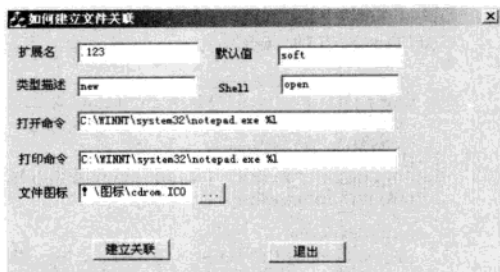


图 7.7 如何建立文件关联

```

::RegSetValue(sub,NULL,REG_SZ,strvalue,strvalue.GetLength());
//创建默认值
::RegCreateKey(HKEY_CLASSES_ROOT,strvalue,&sub);
::RegSetValue(sub,NULL,REG_SZ,strings,strings.GetLength());
//创建DefaultIcon子项
key.Format("%s\\DefaultIcon",strvalue);
::RegCreateKey(HKEY_CLASSES_ROOT,key,&sub);
::RegSetValueEx(sub,NULL,NULL,REG_EXPAND_SZ,(BYTE*)stricon.GetBuffer(0),
 stricon.GetLength());
//创建shell子项
key.Format("%s\\shell",strvalue);
::RegCreateKey(HKEY_CLASSES_ROOT,key,&sub);
//创建shell/open子项
key.Format("%s\\shell\\%",strvalue,strings);
::RegCreateKey(HKEY_CLASSES_ROOT,key,&sub);
//创建shell/open/command子项
key.Format("%s\\shell\\%s\\command",strvalue,strings);
::RegCreateKey(HKEY_CLASSES_ROOT,key,&sub);
::RegSetValueEx(sub,NULL,NULL,REG_EXPAND_SZ,(BYTE*)strcommand.GetBuffer(0),
 strcommand.GetLength());
//创建shell/print/command子项
key.Format("%s\\shell\\%s\\print\\command",strvalue,strings);
::RegCreateKey(HKEY_CLASSES_ROOT,key,&sub);
::RegSetValueEx(sub,NULL,NULL,REG_EXPAND_SZ,(BYTE*)strprint.GetBuffer(0),
 strprint.GetLength());
RegCloseKey(sub);
}

```

(7) “退出”按钮的实现函数代码如下：

```

void CFileRelatDlg::OnExit()
{
 this->OnCancel();
}

```

### 举一反三

根据本实例，读者可以：

- 在右键关联菜单中添加“记事本编辑”命令；
- 在“我的电脑”右键关联菜单中添加新命令。

## 实例 259 控制光驱的自动运行功能

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\07\259

### 实例说明

在 Windows 操作系统中，光盘放入光驱后可自动运行。如果用户不想使光驱自动运行，可通过程序对光驱进行控制。本实例运行后，单击“禁止自动运行”按钮，设置信息将写入到注册表，重新启动计算机后，放入光驱中的光盘将不会自动运行；单击“开始自动运行”按钮，则光盘放入光驱后可以自动运行。程序运行结果如图 7.8 所示。

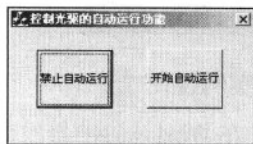


图 7.8 控制光驱的自动运行功能

### 技术要点

本实例应用 RegCreateKey 函数和 RegSetValueEx 函数对注册表进行修改。

打开注册表的 HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\Cdrom 子项，修改 Autorun 的键值数据，设置为 0 可实现禁止光驱自动运行，设置为 1 允许光驱自动运行。

### 实现过程

- 新建一个基于对话框的应用程序。
- 在对话框上添加两个按钮控件，设置 ID 属性分别为 IDC\_FORBID 和 IDC\_START，设置 Caption 属性分别为“禁止自动运行”和“开始自动运行”。



(3) 为“禁止自动运行”按钮添加实现函数 OnForbid, 为“开始自动运行”按钮添加实现函数 OnStart。

(4) “禁止自动运行”按钮的实现函数向注册表中写入数据, 禁止光驱的自动运行, 代码如下:

```
void CCDAutoRunDlg::OnForbid()
{
 DWORD cb;
 HKEY sub;
 DWORD c=0;
 SECURITY_ATTRIBUTES sa;
 sa.nLength = sizeof(SECURITY_ATTRIBUTES);
 sa.bInheritHandle = TRUE;
 sa.lpSecurityDescriptor = NULL;
 CString skey="System\\CurrentControlSet\\Services\\Cdrom";
 ::RegCreateKeyEx(HKEY_LOCAL_MACHINE,skey,0L,"","REG_OPTION_NON_VOLATILE,KEY_ALL_ACCESS,
 &sa,&sub,&cb); //打开注册表
 RegSetValueEx(sub,"Autorun",NULL,REG_DWORD,(BYTE*)&c,sizeof(DWORD)); //禁止光驱自动运行
 RegCloseKey(sub);
}
```

(5) “开始自动运行”按钮的实现函数向注册表中写入数据, 设置光驱的自动运行, 代码如下:

```
void CCDAutoRunDlg::OnStart()
{
 HKEY sub;
 DWORD c=1;
 CString skey="System\\CurrentControlSet\\Services\\Cdrom";
 ::RegCreateKey(HKEY_LOCAL_MACHINE,skey,&sub);
 RegSetValueEx(sub,"Autorun",NULL,REG_DWORD,(BYTE*)&c,sizeof(DWORD)); //设置光驱自动运行
 RegCloseKey(sub);
}
```

### 举一反三

根据本实例, 读者可以:

- 隐藏光驱;
- 设置自动运行光盘。

## 7.4 游戏设置

在 Windows 操作系统中, 自带了许多供用户娱乐的游戏, 但是游戏中的设置不可能满足每一个用户的需求。本节通过几个实例来介绍对游戏进行一些特殊设置的知识。

### 实例 260 设置“蜘蛛纸牌”游戏

这是一个自娱自乐的实例

实例位置: 光盘\mingrisoft\07\260

#### 实例说明

在 Windows XP 操作系统中, 微软公司推出了类似于“空当接龙”的第 3 款排列纸牌游戏“蜘蛛纸牌”。在游戏的过程中, 会带有音乐及动画效果。如果用户想关闭游戏中的声音或动画效果, 可通过本实例实现。本实例运行后, 选中相关选项的复选框, 单击“确定”按钮, 系统重新启动后, 游戏中的声音或动画效果将关闭。运行界面如图 7.9 所示。

#### 技术要点

本实例应用 RegCreateKey 函数和 RegSetValueEx 函数对注册表进行修改。

打开注册表 HKEY\_CURRENT\_USER\Software\Microsoft\Spider 子项, 在 Spider 子项中有

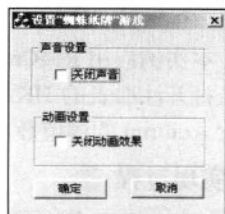


图 7.9 设置“蜘蛛纸牌”游戏

一个名为 Sound 的双字节值项，将该值项的数值数据设置为 0，就可以关闭声音。

在玩“蜘蛛纸牌”游戏的过程中，可以看到在每次发牌时都会有一个动画效果，通过修改注册表，可以关闭该动画效果，打开注册表 HKEY\_CURRENT\_USER\Software\Microsoft\Spider 子项，在 Spider 子项中有一个名为 AnimDeal 的双字节值项，将该值项的数值数据设置为 0 就可以关闭动画效果。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加两个按钮控件，设置 ID 属性分别为 IDC\_BTENTER 和 IDC\_BTEXIT，设置 Caption 属性分别为“确定”和“取消”；添加两个复选框控件，设置 ID 属性分别为 IDC\_CHSOUND 和 IDC\_CHMOVIE，设置 Caption 属性分别为“关闭声音”和“关闭动画效果”，并为两个复选框控件添加成员变量 m\_sound 和 m\_movie；添加两个群组控件，设置 Caption 属性分别为“声音设置”和“动画设置”

(3) 添加“确定”按钮的实现函数 OnEnter；添加“取消”按钮的实现函数 OnExit。

(4) “取消”按钮的实现函数实现关闭窗口体，代码如下：

```
void CSolGameDlg::OnExit()
{
 this->OnCancel();
}
```

(5) “确定”按钮的实现函数向注册表写入信息完成相应的设置，代码如下：

```
void CSolGameDlg::OnEnter()
{
 HKEY sub;
 DWORD val1=1,val2=0;
 if(m_sound.GetCheck())//关闭声音
 {
 CString skey="Software\\Microsoft\\Spider";
 ::RegCreateKey(HKEY_CURRENT_USER,skey,&sub);
 RegSetValueEx(sub,"Sound",NULL,REG_DWORD,(BYTE*)&val2,sizeof(DWORD));
 RegCloseKey(sub);
 }
 else //打开声音
 {
 CString skey="Software\\Microsoft\\Spider";
 ::RegCreateKey(HKEY_CURRENT_USER,skey,&sub);
 RegSetValueEx(sub,"Sound",NULL,REG_DWORD,(BYTE*)&val1,sizeof(DWORD));
 RegCloseKey(sub);
 }
 if(m_movie.GetCheck())//关闭动画
 {
 CString skey="Software\\Microsoft\\Spider";
 ::RegCreateKey(HKEY_CURRENT_USER,skey,&sub);
 RegSetValueEx(sub,"AnimDeal",NULL,REG_DWORD,(BYTE*)&val2,sizeof(DWORD));
 RegCloseKey(sub);
 }
 else //打开动画
 {
 CString skey="Software\\Microsoft\\Spider";
 ::RegCreateKey(HKEY_CURRENT_USER,skey,&sub);
 RegSetValueEx(sub,"AnimDeal",NULL,REG_DWORD,(BYTE*)&val1,sizeof(DWORD));
 RegCloseKey(sub);
 }
}
```

## 举一反三

根据本实例，读者可以：

- 设置“蜘蛛纸牌”窗口的显示位置；
- 关闭“蜘蛛纸牌”的声音。

## 实例 261 修改“扫雷”游戏的设置

这是一个自娱自乐的实例

实例位置: 光盘\mingrisoft\07\261

## 实例说明

Windows 操作系统中自带的“扫雷”游戏可以说是普及率最高的,几乎每个使用过 Windows 的用户都接触过此游戏。如果用户想修改“扫雷”游戏中的原始设置,可通过本实例实现。运行程序,在窗体中对扫雷游戏的相关选项进行设置,单击“确定”按钮后设置信息将写入注册表中,系统重新启动后设置生效。程序运行结果如图 7.10 所示。

## 技术要点

本实例应用 RegCreateKey 函数和 RegSetValueEx 函数对注册表进行修改。

打开注册表 HKEY\_CURRENT\_USER\Software\Microsoft\winmine 子项,在 winmine 子项中有一个名为 Color 的双字节值项,将该值项的数值数据设置为 0,表示不使用颜色,产生黑白效果。

扫雷游戏根据地雷的个数分为初级、中级和高级 3 个级别,如果用户扫雷所使用的时间比记录中的时间短,游戏会提示用户保存新的记录,保存扫雷记录的窗口即扫雷英雄榜。通过修改注册表,可以对扫雷英雄榜进行修改。打开注册表 HKEY\_CURRENT\_USER\Software\Microsoft\winmine 子项,winmine 子项中的字符串值项 Name1、Name2 和 Name3 分别保存了由低到高 3 个级别记录保持者的用户名,双字节值项 Time1、Time2 和 Time3 分别保存了 3 个级别的时间记录。

为了增强扫雷游戏带来的观感刺激,用户可以通过修改注册表,使踩到地雷时产生声音。打开 HKEY\_CURRENT\_USER\Software\Microsoft\winmine 子项,在 winmine 子项中有一个名为 Sound 的双字节值项,将该值项的数值数据设置为 3。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加 2 个按钮控件,设置 ID 属性分别为 IDC\_BTENTER 和 IDC\_BTEXIT,设置 Caption 属性分别为“确定”和“取消”;添加 2 个单选按钮控件,设置 ID 属性分别为 IDC\_BLACK 和 IDC\_COLOR,设置 Caption 属性分别为“黑白界面”和“彩色界面”,并将控件 IDC\_BLACK 的 group 属性选中,添加成员变量 m\_black;添加一个组合框控件,设置 ID 属性为 IDC\_CMGRADE,并添加成员变量 m\_grade;添加 3 个静态文本控件,设置 Caption 属性分别为“级别”、“英雄名称”、“所用时间”;添加两个编辑框控件,设置 ID 属性分别为 IDC\_EDNAME 和 IDC\_EDTIME;添加一个复选框控件,设置 Caption 属性为“引爆地雷发出爆炸声”,并添加成员变量 m\_sound;添加 3 个群组控件,设置 Caption 属性分别为“颜色设置”、“扫雷英雄榜”、“声音设置”。

(3) 添加“确定”按钮的实现函数 OnEnter,添加“取消”按钮的实现函数 OnExit。

(4) 在头文件 WinMineGameDlg.h 中添加变量声明,代码如下:

```
BOOL bcolor;
```

(5) “取消”按钮的实现函数实现窗体的关闭,代码如下:

```
void CWinMineGameDlg::OnExit()
{
 this->OnCancel();
}
```

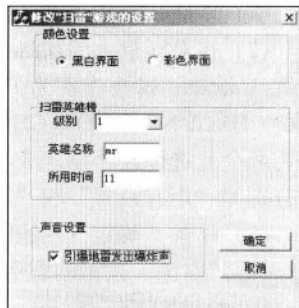


图 7.10 修改“扫雷”游戏的设置

(6) “确定”按钮的实现函数向注册表写入信息完成相应的设置，代码如下：

void CWinMineGameDlg::OnEnter()

```
{
 CString strgrade, strname, strtime;
 m_grade.GetWindowText(strgrade);
 DWORD val1=0; DWORD val2=3;
 DWORD time;
 if(strgrade.IsEmpty())
 {
 AfxMessageBox("请选择级别");
 return;
 }
 GetDlgItem(IDC_EDNAME)->GetWindowText(strname);
 GetDlgItem(IDC_EDTIME)->GetWindowText(strtime);
 time=atoi(strtime);
 HKEY sub;
 CString skey="Software\\Microsoft\\winmine";
 ::RegCreateKey(HKEY_CURRENT_USER, skey, &sub);
 if(m_sound.GetCheck()) //设置声音
 {
 RegSetValueEx(sub, "Sound", NULL, REG_DWORD, (BYTE*)&val2, sizeof(DWORD));
 }
 else
 {
 RegSetValueEx(sub, "Sound", NULL, REG_DWORD, (BYTE*)&val1, sizeof(DWORD));
 }
 if(strgrade.Compare("1")==0) //设置级别1
 {
 RegSetValueEx(sub, "Name1", NULL, REG_SZ, (BYTE*)strname.GetBuffer(strname.GetLength()), strname.GetLength());
 RegSetValueEx(sub, "Time1", NULL, REG_DWORD, (BYTE*)&time, sizeof(DWORD));
 }
 if(strgrade.Compare("2")==0) //设置级别2
 {
 RegSetValueEx(sub, "Name2", NULL, REG_SZ, (BYTE*)strname.GetBuffer(strname.GetLength()), strname.GetLength());
 RegSetValueEx(sub, "Time2", NULL, REG_DWORD, (BYTE*)&time, sizeof(DWORD));
 }
 if(strgrade.Compare("3")==0) //设置级别3
 {
 RegSetValueEx(sub, "Name3", NULL, REG_SZ, (BYTE*)strname.GetBuffer(strname.GetLength()), strname.GetLength());
 RegSetValueEx(sub, "Time3", NULL, REG_DWORD, (BYTE*)&time, sizeof(DWORD));
 }
 if(!m_black.GetCheck()) //设置颜色
 {
 bcolor=TRUE;
 }
 if(bcolor)
 {
 RegSetValueEx(sub, "Color", NULL, REG_DWORD, (BYTE*)&val2, sizeof(DWORD));
 }
 else
 {
 RegSetValueEx(sub, "Color", NULL, REG_DWORD, (BYTE*)&val1, sizeof(DWORD));
 }
 RegCloseKey(sub);
}
```

### 举一反三

根据本实例，读者可以：

- 取消“扫雷”游戏中的“？”标记。

## 7.5 应用软件设置

许多软件在安装的过程中，都会要求用户输入一些注册信息。在软件安装后，如果想修改注册信息，可通过注册表实现。本节将介绍如何设置一些常用应用软件的相关信息。



## 实例 262

设置 Word 2000 文档及  
图片的保存路径

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\07\282

## 实例说明

在默认配置下, Word 文档的保存路径为“我的文档”文件夹, Word 文档中插入图片的打开路径为空。通过修改注册表, 可以对这两项内容进行设置。单击“浏览”按钮选择保存文档的路径和剪贴画路径, 单击“设置”按钮将相关信息写入到注册表, 重新启动计算机后, 设置生效。程序运行结果如图 7.11 所示。



图 7.11 设置 Word 2000 文档及  
图片的保存路径

## 技术要点

本实例应用 RegCreateKey 函数和 RegSetValueEx 函数对注册表进行修改。

打开注册表中的 HKEY\_CURRENT\_USER\Software\Microsoft\Office\X.0\Word\Options 子项, 在 Options 子项中新建一个名为 DOC-PATH 的字符串值项, 该值项的数值数据就是 Word 文档的默认保存路径。

在 Options 子项中新建一个名为 PICTURE-PATH 的字符串值项, 该值项的数值数据就是 Word 文档中插入图片的默认打开路径。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加 2 个静态文本控件, 设置 Caption 属性分别为“设置文档路径”和“剪贴画路径”; 添加 2 个编辑框控件, 设置 ID 属性分别为 IDC\_EDPATH 和 IDC\_EDPICPATH; 添加 3 个按钮, 设置 ID 属性分别为 IDC\_BTTPATH、IDC\_BTTPICPATH 和 IDC\_BTSET, 设置 Caption 属性分别为“浏览”、“浏览”、“设置”。

(3) 添加 ID 属性为 IDC\_BTTPATH 按钮的实现函数 OnBtppath; 添加 ID 属性为 IDC\_BTTPICPATH 按钮的实现函数 OnAddPicPath; 添加 ID 属性为 IDC\_BTSET 按钮的实现函数 OnBtset。

(4) “浏览 (添加文档路径)”按钮的实现函数用来设置文档的保存路径, 代码如下:

```
void CWordPathDlg::OnBtppath()
{
 //创建文件浏览对话框
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer, MAX_PATH);
 bi.hwndOwner=GetSafeHwnd();
 bi.pidlRoot=NULL;
 bi.pszDisplayName=buffer;
 bi.lpszTitle="选择一个文件夹";
 bi.ulFlags=BIF_EDITBOX;
 bi.lpfm=NULL;
 bi.lParam=0;
 bi.ilImage=0;
 LPITEMIDLIST pList=NULL;
 if((pList=SHBrowseForFolder(&bi))!=NULL)
 {
 char path[MAX_PATH];
 ZeroMemory(path, MAX_PATH);
 SHGetPathFromIDList(pList, path);
 GetDlgItem(IDC_EDPATH)->SetWindowText(path);
 }
}
```

(5) “设置”按钮的实现函数向注册表中写入信息完成相应设置, 代码如下:

```
void CWordPathDlg::OnBtset()
{
 HKEY sub;
 CString strpath="",strpicpath="";
 //获得控件中的数据
 GetDlgItem(IDC_EDPATH)->GetWindowText(strpath);
 GetDlgItem(IDC_EDPICPATH)->GetWindowText(strpicpath);
 //向注册表中写入数据
 CString skey="Software\\Microsoft\\Office\\9.0\\Word\\Options";
 ::RegCreateKey(HKEY_CURRENT_USER,skey,&sub);
 RegSetValue(sub,"DOC-PATH",REG_SZ,strpath,strpath.GetLength());
 RegSetValue(sub,"PICTURE-PATH",REG_SZ,strpicpath,strpicpath.GetLength());
 RegCloseKey(sub);
}
```

(6) “浏览 (添加剪贴画)” 按钮的实现函数设置剪贴画位置,代码如下:

```
void CWordPathDlg::OnAddPicPath()
{
 //创建文件浏览对话框
 BROWSEINFO bi;
 char buffer[MAX_PATH];
 ZeroMemory(buffer,MAX_PATH);
 bi.hwndOwner=GetSafeHwnd();
 bi.pidlRoot=NULL;
 bi.pszDisplayName=buffer;
 bi.lpszTitle="选择一个文件夹";
 bi.ulFlags=BIF_EDITBOX;
 bi.lpfnc=NULL;
 bi.lParam=0;
 bi.iImage=0;
 LPITEMIDLIST pList=NULL;
 if((pList=SHBrowseForFolder(&bi))!=NULL)
 {
 char path[MAX_PATH];
 ZeroMemory(path,MAX_PATH);
 SHGetPathFromIDList(pList,path);
 GetDlgItem(IDC_EDPICPATH)->SetWindowText(path);
 }
}
```

### 举一反三

根据本实例,读者可以:

- 设置 Word 文档的日期显示方式;
- 将 Word 中 DOC 格式的表格转换为 HTML 格式的表格;
- 汉化 Word 插入对象的名称。

## 实例 263

### 更改 Photoshop 安装时的 登记信息

本实例是一个人性化的实例

实例位置: 光盘\mingrisoft\07\263

### 实例说明

Photoshop 是一款功能强大的绘图软件,在安装的过程中会要求用户输入一些详细信息,如用户名、公司名称等。在安装完成后,如果用户想对之前的个人信息进行修改,除了通过在注册表中修改外,还可以通过本实例实现此功能。本实例运行后,在各文本框中设置注册信息,单击“确定”按钮,在系统重新启动后,运行 Photoshop 软件时将会显示此注册信息。运行结果如图 7.12 所示。

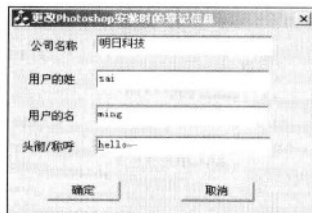


图 7.12 更改 Photoshop 安装时的登记信息

### 技术要点

本实例应用 RegCreateKey 函数和 RegSetValueEx 函数对注册表进行修改。

打开注册表 HKEY\_LOCAL\_MACHINE\Software\Adobe\Registration\User 子项, User 子项中的值项保存了用户安装时的登记信息, 如 COMPAN 值项对应公司名, FNAME 值项对应用户的姓, LNAME 对应用户的名, NAME 值项对应头衔/称呼。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加 4 个静态文本控件, 设置 Caption 属性分别为“公司名称”、“用户的姓”、“用户的名”、“头衔/称呼”; 添加 4 个编辑框控件, 设置 ID 属性分别为 IDC\_EDNAME、IDC\_EDFIRST、IDC\_EDLAST 和 IDC\_EDUSR; 添加 2 个按钮控件, 设置 ID 属性分别为 IDC\_BTENTER 和 IDC\_BTEXIT, 设置 Caption 属性分别为“确定”和“取消”。

(3) 添加“确定”按钮的实现函数 OnEnter; 添加“取消”按钮的实现函数 OnExit。

(4) “取消”按钮的实现函数实现窗体的关闭, 代码如下:

```
void CPhotoShopInfoDlg::OnExit()
{
 this->OnCancel();
}
```

(5) “确定”按钮的实现函数向注册表写入数据完成相应设置, 代码如下:

```
void CPhotoShopInfoDlg::OnEnter()
{
 HKEY sub;
 CString strname, strfirst, strlast, strusr;
 //获得编辑框中数据
 GetDlgItem(IDC_EDNAME)->GetWindowText(strname);
 GetDlgItem(IDC_EDFIRST)->GetWindowText(strfirst);
 GetDlgItem(IDC_EDLAST)->GetWindowText(strlast);
 GetDlgItem(IDC_EDUSR)->GetWindowText(strusr);
 //向注册表中写入数据
 CString skey="Software\\Adobe\\Photoshop\\7.0\\Registration";
 ::RegCreateKey(HKEY_LOCAL_MACHINE,skey,&sub);
 RegSetValueEx(sub,"COMPAN",NULL,REG_SZ,
 (BYTE*)strname.GetBuffer(strname.GetLength()),strname.GetLength());
 RegSetValueEx(sub,"FNAME",NULL,REG_SZ,
 (BYTE*)strfirst.GetBuffer(strfirst.GetLength()),strfirst.GetLength());
 RegSetValueEx(sub,"LNAME",NULL,REG_SZ,
 (BYTE*)strlast.GetBuffer(strlast.GetLength()),strlast.GetLength());
 RegSetValueEx(sub,"NAME",NULL,REG_SZ,
 (BYTE*)strusr.GetBuffer(strusr.GetLength()),strusr.GetLength());
 RegCloseKey(sub);
}
```

## 举一反三

根据本实例, 读者可以:

- 设置 Photoshop 打开和保存图片的路径;
- 设置 Photoshop 中颜色设置的路径。

# 第 8 章

## 数据库技术

- 连接数据库
- 添加数据
- 更新数据
- 删除数据
- 视图
- 存储过程
- 数据库结构的读取与修改
- 图片、多媒体数据录入技术
- 数据备份恢复
- 其他数据库技术

Visual C++



## 8.1 连接数据库

开发数据库应用程序时首先需要连接数据库。Visual C++ 提供了多种连接数据库的方法, 本节将介绍在 Visual C++ 中连接数据库的几种方法。

### 实例 264

### 使用 ODBC DSN 连接 SQL Server 数据库

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\08\264

#### 实例说明

不同的数据库系统, 如 Dbase、FoxPro、Oracle、Informix 甚至是文本文件与 Excel 文件, 都可以通过 ODBC 来达到统一处理的方式, 也就是说通过 ODBC 的驱动程序, 用户可以以一种相同的语法来存取或是维护数据库中的数据。

本实例使用 ODBC DSN 连接 SQL Server 数据库。运行程序前, 首先需要附加 SQL Server 数据库和配置 ODBC 数据源。配置成功后, 运行程序, 即可显示所连接的数据源中的数据, 如图 8.1 所示。

#### 技术要点

为了使客户应用程序能够方便地使用 ODBC, 必须在客户端配置 ODBC 数据源。ODBC 数据源又叫 DSN, 它把客户应用程序所要使用的驱动程序、数据库、用户名和密码等信息组合起来, 供客户端程序使用。

ODBC 数据源有如下 3 种类型。

用户 DSN: 只能由配置该 DSN 的用户使用或只能在当前的计算机上使用。

系统 DSN: 可以被任何使用计算机的人使用。另外, 如果用户要建立 Web 数据库应用程序, 应使用此数据源。

文件 DSN: 除了能够被用户在其他计算机上使用之外, 与系统 DSN 相似。

配置 ODBC 数据源, 可以利用 Windows 控制面板中的 ODBC 数据源管理器来完成, 添加数据源的具体步骤如下。

(1) 单击“控制面板”/“管理工具”, 打开 ODBC 数据源管理器。

(2) 切换到用户 DSN 选项卡, 单击“添加”按钮, 在驱动程序列表中选择 SQL Server, 如图 8.2 所示。

(3) 单击“完成”按钮, 进入添加数据源的设置, 如图 8.3 所示。

在这个对话框中, 必须输入数据源名称, 这个名称是自定义的, 用户可以根据实际情况来设置, 但是需要注意的是: 在程序中调用的名称应与该名称一致。另外, 在“说明”文本框中, 可以对该 ODBC 数据源加以描述, 以便日后维护。在“服务器”下拉列表中选择或输入 SQL 服务器名称。

(4) 单击“下一步”按钮, 弹出如图 8.4 所示的对话框。这里提供了两种登录验证方式: “使用网络登录 ID 的 Windows NT 验证”和“使用用户输入登录 ID 和密码的 SQL 验证”。采用哪种验证方式由系统设置而定, 通常使用 SQL 验证。在“登录 ID”和“密码”文本框中输入用户的名称和密码。

| 编号  | 姓名  | 性别 | 职业  | 年龄 |
|-----|-----|----|-----|----|
| 001 | 李XX | 女  | 服装部 | 25 |
| 002 | 陆X  | 男  | 保安部 | 23 |
| 003 | 李XX | 男  | 保安部 | 22 |
| 004 | 花X  | 男  | 服装部 | 25 |
| 005 | 海X  | 女  | 服装部 | 27 |
| 006 | 王X  | 女  | 服装部 | 21 |
| 007 | 雷X  | 女  | 服装部 | 20 |
| 008 | 李XX | 男  | 销售部 | 26 |
| 009 | 王X  | 男  | 保安部 | 26 |
| 010 | 周X  | 男  | 销售部 | 20 |
| 011 | 周X  | 男  | 销售部 | 22 |
| 012 | 陆X  | 男  | 销售部 | 24 |

图 8.1 使用 ODBC DSN 连接 SQL Server 数据库



图 8.2 创建新数据源

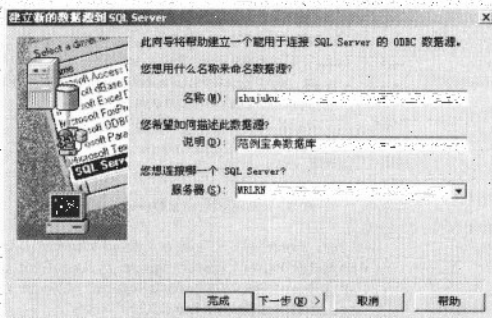


图 8.3 创建到 SQL Server 的新数据源

(5) 单击“下一步”按钮，进入如图 8.5 所示的对话框，在“更改默认的数据库为”下拉列表中选择要连接的数据库名称。

(6) 连续两次单击“下一步”按钮，完成 ODBC 数据源的配置。

(7) 单击“完成”按钮，打开如图 8.6 所示的对话框。可以在该对话框中测试数据源连接是否成功，单击“测试数据源”按钮，如果数据源连接成功，将显示设置成功的信息。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 向窗体中添加一个列表视图控件，为其添加一个成员变量 m\_Grid。

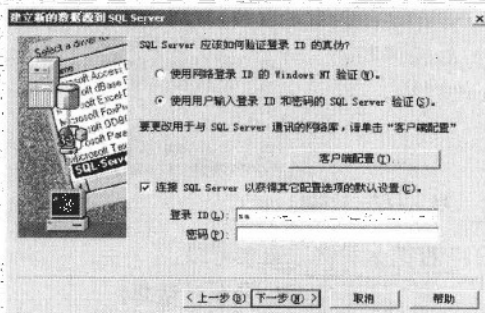


图 8.4 SQL 服务器登录设置

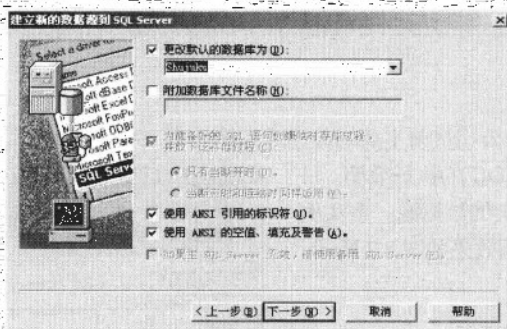


图 8.5 选择数据库名称

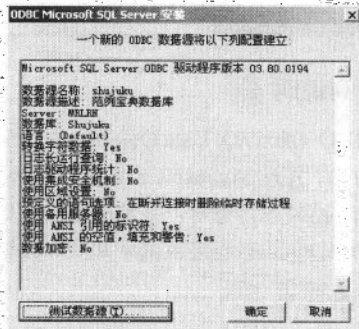


图 8.6 ODBC 设置完成

(3) 主要程序代码如下：

```
m_Grid.SetExtendedStyle(LVS_EX_FLATSB
|LVS_EX_FULLROWSELECT
|LVS_EX_HEADERDRAGDROP
|LVS_EX_ONECLICKACTIVATE
|LVS_EX_GRIDLINES); //设置列表视图控件扩展风格

//设置列标题
m_Grid.InsertColumn(0,"编号",LVCFMT_LEFT,95,0);
m_Grid.InsertColumn(1,"姓名",LVCFMT_LEFT,95,1);
m_Grid.InsertColumn(2,"性别",LVCFMT_LEFT,95,2);
m_Grid.InsertColumn(3,"职业",LVCFMT_LEFT,95,3);
m_Grid.InsertColumn(4,"年龄",LVCFMT_LEFT,95,4);
CString strname;
try
```

```

 {
 strname.Format("Provider=MSDASQL.1;Persist Security Info=False;User ID=sa;\nData Source=shujuku;Initial Catalog=Shujuku"); //设置连接字符串
 m_pConnection.CreateInstance("ADODB.Connection"); //连接对象实例化
 _bstr_t strConnect=strname;
 m_pConnection->Open(strConnect, "", "", adModeUnknown); //连接数据库
 }
 catch(_com_error e)
 {
 AfxMessageBox(e.Description());
 }
 _bstr_t bstrSQL = "select * from kjbdsjk order by 编号 desc"; //设置查询语句
 m_pRecordset.CreateInstance(__uuidof(Recordset)); //记录集对象实例化
 m_pRecordset->Open(bstrSQL, m_pConnection.GetInterfacePtr(), adOpenDynamic,
 adLockOptimistic, adCmdText); //打开记录集
 //向列表视图控件中插入数据
 while(!m_pRecordset->adoEOF)
 {
 m_Grid.InsertItem(0, "");
 m_Grid.SetItemText(0, 0, (char*)(_bstr_t)m_pRecordset->GetCollect("编号"));
 m_Grid.SetItemText(0, 1, (char*)(_bstr_t)m_pRecordset->GetCollect("姓名"));
 m_Grid.SetItemText(0, 2, (char*)(_bstr_t)m_pRecordset->GetCollect("性别"));
 m_Grid.SetItemText(0, 3, (char*)(_bstr_t)m_pRecordset->GetCollect("职业"));
 m_Grid.SetItemText(0, 4, (char*)(_bstr_t)m_pRecordset->GetCollect("年龄"));
 m_pRecordset->MoveNext(); //向下移动记录集指针
 }
 if(m_pRecordset!=NULL)
 m_pRecordset->Close();
 m_pConnection->Close();
}

```

### 举一反三

根据本实例，读者可以：

- 设计 ODBC 登录对话框；
- 在程序中自动配置 ODBC。

## 实例 265 用 ADO 动态连接数据库

本实例可以方便操作、提高效率

实例位置：光盘\mingrisoft\08\265

### 实例说明

ADO (ActiveX Data Objects) 是 Microsoft 公司目前主要的数据库存储技术，ADO 提供了数据库操作方法，并且 Microsoft 公司在不断地改善 ADO 的执行效率。目前 ADO 已经非常稳定，并且能够存取更多种类的数据源。本实例将使用 ADO 动态连接数据库并获取数据。实例效果如图 8.7 所示。

### 技术要点

ADO 是 Microsoft 公司数据库应用程序开发的新接口，是建立在 OLE DB 之上的高层数据库访问技术。ADO 封装了 OLE DB 所提供的接口，使用户能够编写应用程序以通过 OLE DB 提供者访问和操作数据库服务器中的数据。ADO 的优点是易于使用、速度快。ADO 技术不仅可以应用于关系数据库，还可以应用于非关系数据库、电子邮件和文件系统等。可以使用统一的方法对不同的文件系统进行访问，大大简化了程序的编制。

ADO 模型是由 7 个对象所构成的，为了让读者更好地了解 ADO 技术，现将这 7 个 ADO 对象的功能进行简单介绍。

(1) Command 对象。Command 对象定义了对数据源执行的指定命令，它包含对目标数



图 8.7 用 ADO 动态连接数据库



数据库进行某种操作的命令，例如查询数据库、更改数据库结构和参数定义等。

(2) Connection 对象。Connection 对象用于管理与数据库的连接，包括打开连接和关闭连接以及运行 SQL 命令等，它包含了关于目标数据库数据提供者 (Database Provider) 的相关信息。

(3) Recordset 对象。Recordset 对象用于管理来自基本数据库表或 SQL 查询语句执行结果的记录集。通常，Recordset 对象里的所有字段的值指的是数据库当前记录的值。Recordset 对象不仅包含某个查询返回的记录集，还包含记录中的游标 (Cursor)。

(4) Error 对象。Error 对象包含与 ADO 的单个操作 (方法的执行或者属性的读取、赋值) 有关的数据访问错误的详细信息，还包含数据库驱动程序出错时的扩展信息。

(5) Field 对象。Field 对象对应于数据库表的字段或者 SQL 查询语句 SELECT 关键字之后跟着的域，它包含记录集中数据的某单个列的信息。

(6) Parameter 对象。Parameter 对象用于管理基于参数化查询或存储过程的 Command 对象相关联的参数或自变量，这类 Command 对象有一个包含其所有 Parameter 对象的 Parameters 集合。

(7) Property 对象。Property 对象代表由提供者定义的 ADO 对象的动态特征。


在与数据源进行通信之前，必须先与它建立连接关系。数据库操作结束之后，还要关闭此连接。这种打开和关闭一个连接的操作与打电话的过程有相似之处。

下面主要介绍 Connection 对象。

Connection 对象代表与数据源进行的通信会话。如果是客户端/服务器数据库系统，则 Connection 对象等价于到服务器的实际网络连接。因为某些提供者可能会不支持某些数据库服务器功能 (批更新的)，所以 Connection 对象的某些集合、方法或属性有可能无效。

使用 Connection 对象的集合、方法和属性可执行下列操作。

- 在打开连接前使用ConnectionString、ConnectionTimeout 和 Mode 属性对连接进行配置。
- 设置 CursorLocation 属性以便调用支持批更新的“客户端游标提供者”。
- 使用 DefaultDatabase 属性设置连接的默认数据库。
- 使用 IsolationLevel 属性为在连接上打开的事务设置隔离级别。
- 使用 Provider 属性指定 OLE DB 提供者。
- 使用 Open 方法建立到数据源的物理连接，使用 Close 方法将其切断。
- 使用 Execute 方法执行对连接的命令，并使用 CommandTimeout 属性对执行进行配置。
- 可使用 BeginTrans、CommitTrans 和 RollbackTrans 方法以及 Attributes 属性管理打开的连接上的事务 (如果提供者支持则包括嵌套的事务)。
- 使用 Errors 集合检查数据源返回的错误。
- 通过 Version 属性读取所使用的 ADO 执行版本。
- 使用 OpenSchema 方法获取数据库纲要信息。

 注意：如果不使用 Command 对象执行查询，向 Connection 对象的 Execute 方法传送查询字符串。当需要使命令文本具有持久性并重新执行或使用查询参数的时候，则必须使用 Command 对象。

Connection 对象属性如表 8.1 所示。

表 8.1 Connection 对象属性表

| 属 性              | 描 述                                                                   |
|------------------|-----------------------------------------------------------------------|
| Attributes       | 其值可以为 AdXactCommitRetaining 和 AdXactAbortRetaining 中的任意一个或多个          |
| CommandTimeout   | 该属性允许由于网络拥塞或服务器负载过重产生的延迟而取消 Execute 方法调用，指示在终止尝试和产生错误之前执行命令期间需等待的时间   |
| ConnectionString | 该属性包含用来建立到数据源的连接的信息。通过传递包含一系列由分号分隔的 argument = value 语句的详细连接字符串可指定数据源 |



续表

| 属 性               | 描 述                                                                         |
|-------------------|-----------------------------------------------------------------------------|
| ConnectionTimeout | 如果由于网络拥塞或服务器负载过重导致的延迟使得必须放弃连接尝试时，使用该属性，指示在终止尝试和产生错误前建立连接期间所等待的时间            |
| CursorLocation    | 该属性允许在可用于提供者的各种游标库中进行选择，通常，可以选择使用客户端游标库或位于服务器上的某个游标库，设置或返回游标引擎的位置           |
| DefaultDatabase   | 使用 DefaultDatabase 属性可设置或返回指定 Connection 对象上默认数据库的名称，指示 Connection 对象的默认数据库 |
| IsolationLevel    | 表示 Connection 对象的隔离级别，IsolationLevel 的属性为读/写，直到下次调用 BeginTrans 方法时，该设置才可以生效 |
| Mode              | 可设置或返回当前连接上提供者正在使用的访问权限，Mode 属性只能在关闭 Connection 对象时设置                       |
| Provider          | 可设置或返回连接提供者的名称                                                              |
| State             | 可以随时使用 State 属性确定指定对象的当前状态，该属性是只读的                                          |
| Version           | 表示 ADO 版本号                                                                  |

Connection 对象方法如表 8.2 所示。

表 8.2 Connection 对象方法表

| 方 法           | 描 述                             |
|---------------|---------------------------------|
| BeginTrans    | 开始一个新事务                         |
| CommitTrans   | 保存所有更改并结束当前事务，也可以启动新事务          |
| RollbackTrans | 取消当前事务中所做的任何更改并结束事务，也可以启动新事务    |
| Cancel        | 取消执行挂起的异步 Execute 或 Open 方法的调用  |
| Close         | 关闭打开的对象及任何相关对象                  |
| Execute       | 执行指定的查询、SQL 语句、存储过程或特定提供者的文本等内容 |
| Open          | 打开到数据源的连接                       |
| OpenSchema    | 从提供者获取数据库纲要信息                   |

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 向窗体中添加一个编辑框控件、一个列表框控件和一个按钮控件，为编辑框添加变量 m\_edit，为列表框控件添加成员变量 m\_list。

(3) 主要程序代码如下：

```
void CADatabaseDlg::OnButconnect()
{
 UpdateData(true);
 m_list.ResetContent();
 OnInitADOConn(m_edit);
 _bstr_t bstrSQL;
 bstrSQL = "select*from sysobjects where xtype='U'"; //设置查询语句
 CString strText;
 m_pRecordset.CreateInstance(__uuidof(Recordset)); //记录集对象实例化
 m_pRecordset->Open(bstrSQL, m_pConnection.GetInterfacePtr(), adOpenDynamic,
 adLockOptimistic, adCmdText); //打开记录集
 //向列表框控件中插入数据
 while(m_pRecordset->adoEOF==0)
 {
 strText=(char*)(_bstr_t)m_pRecordset->GetCollect("name"); //获得数据库中数据
 m_list.AddString(strText);
 m_pRecordset->MoveNext(); //向下移动记录集指针
 }
 ExitConnect();
}
//连接数据库
void CADatabaseDlg::OnInitADOConn(CString strsjk)
```

```

{
 ::CoInitialize(NULL);
 CString strname;
 strname.Format("Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;\
 Initial Catalog=%s;Data Source=",strsjk); //设置连接字符串
 try
 {
 m_pConnection.CreateInstance("ADODB.Connection"); //连接对象实例化
 _bstr_t strConnect=strname;
 m_pConnection->Open(strConnect, "", "", adModeUnknown); //连接数据库
 }
 catch(_com_error e)
 {
 AfxMessageBox(e.Description());
 }
}
//断开
void CADatabaseDlg::ExitConnect()
{
 if(m_pRecordset!=NULL)
 m_pRecordset->Close();
 m_pConnection->Close();
}
}

```

### 举一反三

根据本实例，读者可以：

- 动态连接 Access 数据库；
- 连接加密的 Access 数据库。

## 8.2 添加数据

SQL 中提供了 INSERT 命令和 SELECT INTO 命令作为添加数据的主要方法。其中最基本的形式是简单地插入一行数据，最复杂的形式则是能够使用一个复杂的 SELECT 语句创建数据集，并利用这个数据集来创建一个表。本节将通过几个实例介绍在 SQL 中利用 INSERT 命令或 SELECT INTO 命令添加数据方面的知识。

### 实例 266

#### 利用 INSERT 语句批量插入数据

这是一个可以启发思维的实例

实例位置：光盘\mingrisoft\08\266

### 实例说明

开发应用程序时，有时需要批量向数据表中插入数据（如根据需向临时数据表中插入数据）。运行程序，单击“确定”按钮，即可将员工表中所有记录的编号和姓名插入到另一个表中，如图 8.8 所示。

### 技术要点

本节使用最简单、最直接的方法（INSERT/VALUES 语句实现的方法）向数据表中插入数据。这种方法只能接收一组值，因此它只能一次插入一行数据。将 INSERT 语句放置在循环语句当中，可实现向数据表中批量插入数据的功能。

INSERT INTO 语法格式如下：



图 8.8 利用 INSERT 语句批量插入数据

```
INSERT [INTO] owner Table [(columns,...)]
VALUES(value,...)
```

参数说明:

- [INTO]: 可选择的关键字。
- owner Table: 表名或视图名。
- columns: 要在其中插入数据的一列或多列的列表, 必须用圆括号括起来, 用逗号进行分隔。
- value: 引入需要的数据值的列表, column\_list 或者表中的每一列都必须有一个值。

## 实现过程

(1) 创建一个基于对话框的应用程序。

(2) 在菜单中选择 Project/Add To Project/Components and Controls 命令, 弹出 Components and Controls Gallery 窗口, 双击窗口中的 Registered ActiveX Controls 文件夹, 双击 Microsoft ADO Data Control6.0(SP4)(OLEDB)选项, 取默认值, 添加控件, 单击“Close”按钮, ADO Data 控件就添加到控件面板中了, 用同样的方法找到 Microsoft DataGrid Control6.0(SP5)(OLEDB)选项, 添加 DataGrid 控件。

(3) 在窗体上添加两个 ADO Data 控件和两个 DataGrid 控件, 为 ADO Data 控件添加变量 m\_adoc。

(4) 创建一个 ADOConn 类, 用于连接数据库。

(5) 主要程序代码如下:

```
void CInsertdatesDlg::OnOK()
{
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn(); //连接数据库
 _bstr_t vSQL,sql;
 CString str1,str2;
 vSQL="select*from kjbdsjk order by 编号"; //设置查询语句
 _RecordsetPtr m_pRecordset;
 m_pRecordset=m_AdoConn.GetRecordSet(vSQL); //打开记录集
 while(m_pRecordset->adoEOF==0)
 {
 //获得数据库中数据
 str1=(char*)(_bstr_t)m_pRecordset->GetCollect("编号");
 str2=(char*)(_bstr_t)m_pRecordset->GetCollect("姓名");
 //设置插入语句
 sql="insert into inserttable (编号,姓名) values('"+str1+"','"+str2+"')";
 m_AdoConn.ExecuteSQL(sql);
 m_pRecordset->MoveNext();
 }
 m_AdoConn.ExitConnect();
 m_adoc.SetRecordSource("select*from inserttable");
 m_adoc.Refresh();
 //CDialog::OnOK();
}
```

## 举一反三

根据本实例, 读者可以:

- 利用 INSERT 语句向表中添加数据。

## 实例 267

## 利用 SELECT INTO 生成临时表

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\08\267

## 实例说明

在应用程序中有时需要将数据信息保存到临时表中, 以免数据丢失。本实例实现的是当程序运行的时候单击窗体中的“生成临时表”按钮, 根据员工表中的编号、姓名和年龄字段创建

一个临时表，如图 8.9 所示。

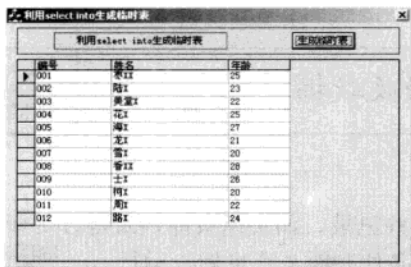


图 8.9 利用 SELECT INTO 生成临时表

## 技术要点

使用 SELECT 语句的 INTO 选项可以创建一个临时表来包含 SELECT 语句的结果集。在数据转换期间以及在必须动态地处理可变的源表结构的应用程序中，经常会用到 SELECT INTO 命令。

SELECT INTO 命令的简明语法如下：

```
SELECT Columns
INTO NewTable
FROM DataSourcees
[WHERE Conditions]
```

参数说明：

- Columns：字段列表。
- NewTable：创建的临时表的表名。
- DataSourcees：数据源表。
- Conditions：WHERE 语句的条件。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个 ADO Data 控件和一个 DataGrid 控件，为 ADO Data 控件添加变量 m\_adoc.
- (3) 创建一个 ADOConn 类，用于连接数据库。
- (4) 主要程序代码如下：

```
void CSelectintoDlg::OnOK()
{
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn(); //连接数据库
 _bstr_t vSQL;
 vSQL="select 编号,姓名,年龄 into linshibiao from kjbdsjk"; //设置SQL语句
 m_AdoConn.ExecuteSQL(vSQL); //生成临时表
 m_AdoConn.ExitConnect();
 m_adoc.SetRecordSource("select * from linshibiao"); //设置临时表为控件的数据源
 m_adoc.Refresh(); //刷新数据源
 //CDialog::OnOK();
}
```

## 举一反三

根据本实例，读者可以：

- 将操作软件的日志信息保存在临时表中。

## 8.3 更新数据

本节通过几个实例介绍一下运用 SQL 语句 (UPDATE 语句) 更新数据的方法，使读者更加



深刻地掌握利用 SQL 语句更新数据库中数据的相关知识。

## 实例 268 批量修改数据

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\08\268

### 实例说明

有时候需要批量更新一些记录, 如果逐条地判断是否更新, 工作起来非常麻烦, 这时就要指定一定的条件, 根据条件判断是否更新。运行程序, 在文本框中输入相关的数据信息批量修改数据, 图 8.10 所示的是将员工表中工资低于 1 600 的工资都修改为 3 000。

### 技术要点

使用 UPDATE 语句可以修改数据表中的记录。

UPDATE 语句的语法格式如下:

```
UPDATE<table name | view name>
SET <column name>=<expression>
[WHERE<search condition>]
```

参数说明:

- <table name | view name>: 表名或视图名。
- column name: 含有要更改数据的列的名称。
- expression: 变量、表达式或加上括号的返回单个值的自查询语句。
- search condition: WHERE 语句的条件。

在 UPDATE 语句中加入 WHERE 条件可以实现批量修改数据的功能, 如在本实例中, 在 UPDATE 语句 WHERE 条件中添加“工资<=1 600”语句就可以修改工资小于等于 1 600 的记录信息, 本实例实现的语法如下:

```
UPDATE kjbdsjk SET 工资=3000 WHERE 工资<=1600
```

### 实现过程

(1) 新建一个基于对话框的应用程序, 将窗体标题改为“批量修改数据”。

(2) 在窗体上添加两个编辑框控件、一个 ADO Data 控件和一个 DataGrid 控件, 为编辑框添加变量, 为 ADO Data 控件添加变量 m\_adodc。

(3) 创建一个 ADOConn 类, 用于连接数据库。

(4) 主要程序代码如下:

```
void CUpdatedatasDlg::OnOK()
{
 UpdateData(true);
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn(); //连接数据库
 _bstr_t sql;
 CString str1,str2;
 //格式化编辑框中数据
 str1.Format("%d",m_laborage1);
 str2.Format("%d",m_laborage2);
 sql = "update kjbdsjk set 工资="+str2+" where 工资 <="+str1+" "; //设置修改语句
 m_AdoConn.ExecuteSQL(sql); //执行修改语句
 m_AdoConn.ExitConnect();
 m_adodc.SetRecordSource("select 编号,姓名,职业,工资 from kjbdsjk"); //设置控件数据源
 m_adodc.Refresh(); //刷新数据源
 //CDialog::OnOK();
}
```



图 8.10 批量修改数据

## 举一反三

根据本实例，读者可以：

- 修改指定条件的数据信息。

## 实例 269

将指定字段数据为空的记录  
添上数据

本实例可以提高工作效率

实例位置：光盘\mingrisoft\08\269

## 实例说明

在编写应用程序的时候，可以将数据表中的某些数据信息先设置为空值，以便以后用到时再添加，如在本实例的员工工数据表中添加数据信息时，可以先不添加员工的职业，等分配好工作后再根据员工姓名添加职业，如图 8.11 所示。



图 8.11 将指定字段数据为空的记录添上数据

## 技术要点

利用 UPDATE 可以实现向记录集中添加数据信息的功能，本实例中，在“路 X”的职业中填写“销售部”信息，实现的语句如下：

```
UPDATE kongbiao SET 职业='销售部' WHERE 姓名='路X'
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加两个编辑框控件、一个 ADO Data 控件和一个 DataGrid 控件，为编辑框添加变量，为 ADO Data 控件添加变量 m\_adodc。
- (3) 创建一个 ADOConn 类，用于连接数据库。
- (4) 主要程序代码如下：

```
void CUpdateemptyDlg::OnOK()
{
 UpdateData(true);
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn(); //连接数据库
 _bstr_t sql;
 sql = "update kongbiao set 职业='"+m_work+"' where 姓名='"+m_name+"' ";
 m_AdoConn.ExecuteSQL(sql); //将空字段添加指定数据
 m_AdoConn.ExitConnect();
 m_adodc.SetRecordSource("select*from kongbiao");
 m_adodc.Refresh();
 //CDialog::OnOK();
}
```

## 举一反三

根据本实例，读者可以：

- 在数据表记录中添加系统当前日期；
- 根据员工的状态填写员工是否离职的信息。

## 8.4 删除数据

运用 DELETE 语句可以删除选定的数据记录。DELETE 语句后的 FROM 子句指定了包含记录的目标表。WHERE 子句则指定了表中被删除记录的条件，本节中通过几个实例介绍运用 DELETE 语句删除数据信息的相关知识。

## 实例 270 删除单条数据

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\08\270

## 实例说明

利用 DELETE 语句可以删除数据表中的数据, 如本实例中, 在窗体的文本框中输入要删除员工的姓名, 单击“删除”按钮, 该员工的基本信息将被删除, 如图 8.12 所示。

## 技术要点

运用 DELETE 语句实现删除数据记录。

DELETE 语句的语法格式如下:

```
DELETE FROM <table name>
[WHERE<search condition>]
```

参数说明:

- table name: 表名或视图名。
- search condition: WHERE 语句的条件。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个编辑框控件、一个 ADO Data 控件和一个 DataGrid 控件, 为编辑框添加变量, 为 ADO Data 控件添加变量 m\_adodc。
- (3) 创建一个 ADOConn 类, 用于连接数据库。
- (4) 主要程序代码如下:

```
void CDeletedataDlg::OnOK()
{
 UpdateData(true);
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn();
 _bstr_t sql;
 sql = "delete from kjbdsjk where 姓名='"+m_name+"'"; //设置删除语句
 m_AdoConn.ExecuteSQL(sql); //删除指定条件的数据
 m_AdoConn.ExitConnect();
 m_adodc.SetRecordSource("select*from kjbdsjk");
 m_adodc.Refresh();
 //CDialog::OnOK();
}
```

## 举一反三

根据本实例, 读者可以:

- 删除指定条件下的数据信息;
- 删除指定相同条件的多条记录信息;
- 删除数据表中所有的数据信息。

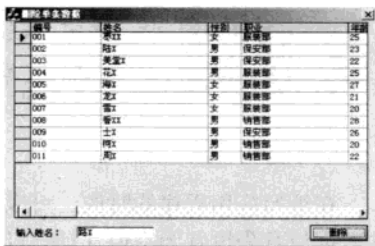


图 8.12 删除单条数据

## 实例 271 删除数据库中无用处的记录

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\08\271

## 实例说明

数据表中如果存在某些无用处的记录信息, 应该将其删除。本示例演示的是当程序运行时在窗体的数据网格中显示离职员工和没有离职的员工的所有信息, 当单击“删除”按钮之后,

数据表中的离职员工的信息被删除,如图 8.13 所示。

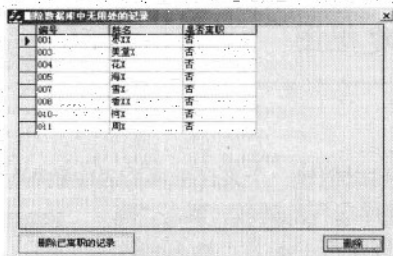


图 8.13 删除数据库中无用处的记录

## 技术要点

将 DELETE 语句后的 WHERE 条件设置为“是否离职=‘是’”,即可将数据表中所有离职员工的信息删除,本实例实现的语句如下:

```
DELETE FROM tb_kongbiao WHERE 是否离职='是'
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个 ADO Data 控件和一个 DataGrid 控件,为 ADO Data 控件添加变量 m\_adodc。
- (3) 创建一个 ADOConn 类,用于连接数据库。
- (4) 主要程序代码如下:

```
void CDeletenodeDlg::OnOK()
{
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn();
 _bstr_t vSQL,sql;
 CString str;
 vSQL="select*from kongbiao order by 编号";
 _RecordsetPtr m_pRecordset;
 m_pRecordset=m_AdoConn.GetRecordSet(vSQL);
 while(!m_pRecordset->adoEOF)
 {
 str=(char*)(_bstr_t)m_AdoConn.m_pRecordset->GetCollect("是否离职"); //获得指定自动数据
 if(str=="是") //判断是否符合删除条件
 {
 sql="delete from kongbiao where 是否离职='"+str+"'"; //设置删除字符串
 m_AdoConn.ExecuteSQL(sql); //删除语句
 }
 m_pRecordset->MoveNext();
 }
 m_AdoConn.ExitConnect();
 m_adodc.SetRecordSource("select 编号,姓名,是否离职 from kongbiao");
 m_adodc.Refresh();
 //CDialog::OnOK();
}
```

## 举一反三

根据本实例,读者可以:

- 删除指定数据表中的所有空记录信息。

## 8.5 视图

视图是存在于数据库中的“虚表”,视图的内容由查询语句定义。对数据库用户来讲,视图似乎是一张真正的表,具有一组命名的数据记录和字段。但是,与真实的表不同,视图不会像一组实际存储的数据那样存储在数据库中。相反,视图中可见的字段和记录是由定义该视图的查询直接生成查询结果。本节将通过几个实例介绍关于视图方面的知识。



## 实例 272 动态创建视图

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\08\272

## 实例说明

利用 CREATE VIEW 语句可以创建一个来自于多个表中数据信息的视图，使数据表中的数据相互关联。本实例实现的是当程序运行的时候，单击“确定”按钮之后，动态创建一个包含员工编号、姓名和收入信息的视图 view\_name，并将该视图中的数据显示在窗体的数据网格当中，如图 8.14 所示。

## 技术要点

利用 CREATE VIEW 语句可以创建视图，使数据表中的数据相互关联。

CREATE VIEW 语句的语法格式如下：

```
CREATE VIEW
view_name[(column_name[,column_name]...)]
AS
select_statement
```

参数说明：

- view\_name：视图的名称，视图名称必须符合标识符规则。
- column\_name：视图中的列名。
- AS：视图要执行的操作。
- select\_statement：定义视图的 SELECT 语句。该语句可以使用多个表或其他视图，若要从创建视图的 SELECT 子句所引用的对象中选择，必须具有适当的权限。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个 ADO Data 控件和一个 DataGrid 控件，为 ADO Data 控件添加变量 m\_adodc。

- (3) 创建一个 ADOConn 类，用于连接数据库。

- (4) 主要程序代码如下：

```
void CDtcjviewDlg::OnOK()
{
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn();
 _bstr_t sql;
 sql="create view view_name as select k.编号,k.姓名,d.收入 from kjbdsjk as k inner join duobiao \
 as d on k.编号 = d.编号 where ((k.编号)=[d].[编号])"; //设置创建视图语句
 m_AdoConn.ExecuteSQL(sql); //创建视图
 m_AdoConn.ExitConnect();
 m_adodc.SetRecordSource("select*from view_name");
 m_adodc.Refresh();
 //Dialog::OnOK();
}
```

## 举一反三

根据本实例，读者可以：

- 动态创建来自多表数据的复杂视图；
- 在创建视图后向视图中插入数据。



图 8.14 动态创建视图

## 实例 273 通过视图更改数据

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\08\273

## 实例说明

视图关联着多个表, 因此更改视图中的数据同时也更改了与其相关联的数据表中的数据。在编写数据库程序时运用视图可以更改多个表中的数据, 从而减少应用程序中的代码量, 提高程序运行的效率和程序编写的速度。运行程序, 在“职业”文本框中输入“保安部”, 在“奖金”文本框中输入“500”, 单击“保存”按钮, 将把保安部的奖金都改为 500, 如图 8.15 所示。

| 姓名  | 职业  | 奖金   |
|-----|-----|------|
| 001 | 保安部 | 1000 |
| 002 | 保安部 | 500  |
| 003 | 保安部 | 500  |
| 004 | 保安部 | 250  |
| 005 | 保安部 | 420  |
| 006 | 保安部 | 350  |
| 007 | 保安部 | 127  |
| 008 | 销售部 | 1000 |
| 009 | 销售部 | 500  |
| 010 | 销售部 | 1300 |
| 011 | 销售部 | 1021 |
| 012 | 销售部 | 525  |

图 8.15 通过视图更改数据

## 技术要点

本实例的实现语句如下:

```
UPDATE view_jiangjin SET 奖金=500 WHERE 职业='保安部'
```

该语句将把所有保安部的人的奖金都改为 500。

注意: 当通过 SQL 语句修改视图中数据的时候, 与其相关联数据表中的数据也同时被修改。

## 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“通过视图更改数据”。
- (2) 在窗体上添加两个编辑框控件、一个 ADO Data 控件和一个 DataGrid 控件, 为 ADO Data 控件添加变量 m\_adodc。
- (3) 创建一个 ADOConn 类, 用于连接数据库。
- (4) 主要程序代码如下:

```
void CModviewDlg::OnOK()
{
 UpdateData(true);
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn();
 _bstr_t sql;
 CString str;
 str.Format("%d", m_money);
 sql="update view_jiangjin set 奖金="+str+" where 职业='"+m_work+"'";//设置修改语句
 m_AdoConn.ExecuteSQL(sql); //修改语句
 m_AdoConn.ExitConnect();
 m_adodc.SetRecordSource("select*from view_jiangjin");
 m_adodc.Refresh();
 //CDialog::OnOK();
}
```

## 举一反三

根据本实例, 读者可以:

- 更新来源于多个表中的数据;
- 更新具有复杂条件视图中的数据。

## 实例 274 删除视图

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\08\274

## 实例说明

有时根据应用程序的需求要建立一些临时的视图, 当这些视图不需要的时候, 就可以将其

删除,避免占用数据库空间。本实例实现的是如何删除数据库中已经存在的视图,运行程序,在窗体的列表框中将显示数据库中所有的视图信息,选中列表中的一个视图之后,单击“确定”按钮,将所选择的视图删除,如图 8.16 所示。

### 技术要点

利用 DROP VIEW 语句可以实现删除数据库中已存在的视图。DROP VIEW 的语法格式如下:

DROP VIEW view\_name

参数说明:

● view\_name: 要删除的视图名称,视图名称必须符合标识符规则。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个列表框控件,为列表框添加成员变量 m\_list。
- (3) 主要程序代码如下:

```
void CDelviewDlg::OnOK()
{
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn();
 _bstr_t sql;
 CString str;
 m_list.GetText(m_list.GetCurSel(),str);
 sql="drop view "+str+" ";
 m_AdoConn.ExecuteSQL(sql);
 m_list.ResetContent();
 _bstr_t vSQL;
 vSQL="select*from sysobjects where xtype='V'"; //查询视图
 _RecordsetPtr m_pRecordset;
 m_pRecordset=m_AdoConn.GetRecordSet(vSQL);
 while(m_pRecordset->adoEOF==0)
 {
 str=(char*)(_bstr_t)m_pRecordset->GetCollect("name");//获得视图名
 m_list.AddString(str); //插入到列表框中
 m_pRecordset->MoveNext(); //向下移动记录集指针
 }
 m_AdoConn.ExitConnect();
 CDialog::OnOK();
}
```



图 8.16 删除视图

### 举一反三

根据本实例,读者可以:

- 删除数据库应用程序中的过时视图。

## 8.6 存储过程

存储过程可以理解为将常用的或很复杂的工作预先以 SQL 程序写好,然后指定一个程序名称保存起来,那么以后只要使用 EXECUTE 指令来执行这个程序,就可自动完成该项工作。本节将通过几个实例介绍有关存储过程方面的知识。

### 实例 275 创建存储过程

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\08\275

### 实例说明

在开发应用程序时,根据需要可以动态地创建存储过程。本实例实现的是如何在 SQL Server

数据库中创建存储过程。运行程序，在“输入存储过程名称”文本框中输入所要创建存储过程的名称，然后在“输入存储过程语法结构”文本框中输入创建存储过程的 SQL 语句，单击“创建存储过程”按钮，完成创建存储过程的操作，此时在数据库中就可以看到被创建的存储过程，如图 8.17 所示。

## 技术要点

可以利用 CREATE PROCEDURE 语句创建存储过程。语法格式如下：

```
CREATE PROC[EDURE] Procedure_name
[.number] [@Parameter data_type [VARYING] [=default] [OUTPUT]]
[...n1]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION }]
[FOR REPLICATION]
AS sql_statement [...n2]
```

参数说明：

- Procedure\_name：用于指定存储过程名，必须符合标识符规则，且对于数据库及其所有者必须惟一。创建局部临时存储过程，可以在 Procedure\_name 前面加一个“#”；创建全局临时存储过程，可以在 Procedure\_name 前面加“##”。
- number：为可选的整数，用于区分同名的存储过程，以使用一条 DROP PROCEDURE 语句删除一组存储过程。
- @Parameter：是存储过程的形参，“@”符号作为第一个字符来指定参数名称，必须符合标识符规则。创建存储过程时，可以声明一个或多个参数，执行存储过程时应提供响应的实在参数，除非定义了该参数的默认值，默认参数值只能是常量。
- data\_type：用于指定形参的数据类型，形参可以是 SQL Server 支持的任何类型，但 cursor 类型只能用于 OUTPUT 参数，同时必须指定 VARYING 和 OUTPUT 关键字。
- default：指定存储过程输入参数的默认值，必须是常量或 NULL，默认值中可以包含通配符。
- n1：表示存储过程指定的参数个数。
- RECOMPILE：表示 SQL Server 每次运行时对其重新编译。
- ENCRYPTION：表示对存储过程加密。
- FOR REPLICATION：用于说明不能在订阅服务器上执行复制、创建的存储过程，该项不能和 WITH RECOMPILE 同时使用。
- sql\_statement：表示过程中包含的 SQL 语句。
- n2：表示包含 SQL 语句的个数。

本实例创建的存储过程 qqqq 如图 8.18 所示。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加两个编辑框控件、一个 ADO Data 控件和一个 DataGrid 控件，为 ADO Data 控件添加变量 m\_adodc。
- (3) 创建一个 ADOConn 类，用于连接数据库。
- (4) 主要程序代码如下：

```
void CSavecourseDlg::OnOK()
{
 UpdateData(true);
```

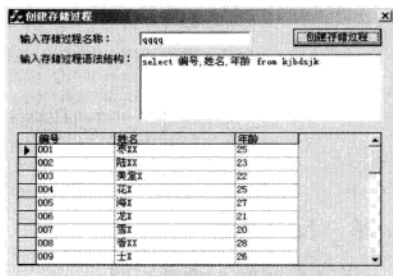


图 8.17 创建存储过程

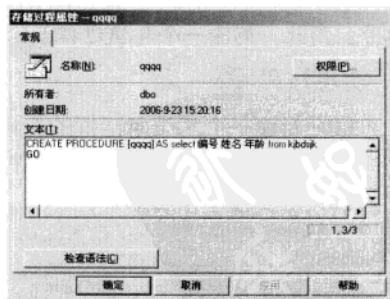


图 8.18 本实例创建的存储过程



```

ADOConn m_AdoConn;
m_AdoConn.OnInitADOConn();
_bstr_t sql;
sql = "CREATE PROCEDURE [" + m_edit1 + "] AS " + m_edit2 + " "; //设置创建存储过程的语句
m_AdoConn.ExecuteSQL(sql); //执行创建语句
m_AdoConn.ExitConnect();
m_adocdc.SetCommandType(4); //设置控件数据源类型
m_adocdc.SetRecordSource(" " + m_edit1 + " "); //设置数据源
m_adocdc.Refresh();
//CDialog::OnOK();
}

```

### 举一反三

根据本实例，读者可以：

- 创建具有复杂结构的存储过程；
- 创建涉及多个表的存储过程。

## 实例 276 删除存储过程

本实例可以提高工作效率

实例位置：光盘\mingrisoft\08\276

### 实例说明

存储过程在应用完成之后，为了节省数据空间，可以将其删除。本实例实现的是如何删除 SQL Server 数据库中已经存在的存储过程，在窗体的文本框中输入要删除数据库中存储过程的名称之后，单击“删除”按钮将存储过程删除，如图 8.19 所示。

### 技术要点

删除存储过程的语句结构如下：

```
DROP PROCEDURE <Procedure name>
```

参数说明：

- Procedure name：存储过程名。

**注意：**在删除一个存储过程时，必须同时删除存储过程的所有版本。在同一 DBMS 内，可以创建同一个存储过程的多个版本。虽然在执行 ALTER PROCEDURE 或 EXEC 语句时，通过提供版本号可以修改并执行某个特定版本的存储过程，但 DROP PROCEDURE 语句不允许输入版本号。因此，删除一个存储过程就意味着删除所有版本。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个编辑框控件，为编辑框控件添加变量 m\_edit。
- (3) 主要程序代码如下：

```

void CDelsaveDlg::OnOK()
{
 UpdateData(true);
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn();
 _bstr_t sql;
 sql = "drop procedure " + m_edit + " "; //设置删除存储过程的语句
 m_AdoConn.ExecuteSQL(sql); //执行删除语句
 m_AdoConn.ExitConnect();
 MessageBox("存储过程已删除！");
 CDialog::OnOK();
}

```

### 举一反三

根据本实例，读者可以：

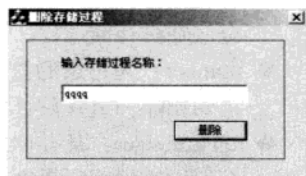


图 8.19 删除存储过程

- 在执行应用程序时删除存储过程;
- 在程序代码中删除指定的存储过程。

## 实例 277 在程序中使用存储过程

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\08\277

### 实例说明

存储过程与触发器一样,在应用程序中经常会用到。本实例演示的是在应用程序中如何使用存储过程。运行程序,在窗体的数据网格中将显示存储过程中的数据信息,如图 8.20 所示。

### 技术要点

通过 Visual C++ 中的 ADO Data 控件可以连接设计好的存储过程,连接的方法如下所述。

(1) 将 ADO Data 控件添加到窗体上,在 ADO Data 控件上单击鼠标右键,在弹出的快捷菜单中选择 Properties 菜单项,打开属性窗口。在该窗口中选择 Control 选项卡,如图 8.21 所示。



图 8.20 在程序中使用存储过程

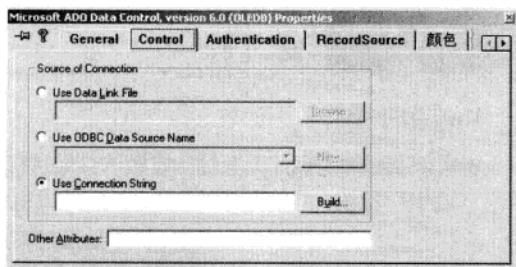


图 8.21 属性页对话框

(2) 选中“Use Connection String”单选按钮,单击“Build”按钮,弹出数据链接属性对话框,在提供者列表中选择 Microsoft OLE DB Provider for SQL Server 选项,如图 8.22 所示。

(3) 单击“下一步”按钮,在连接选项卡中设置对 SQL Server 的连接属性,包括选择服务器、选择数据库,具体如图 8.23 所示。

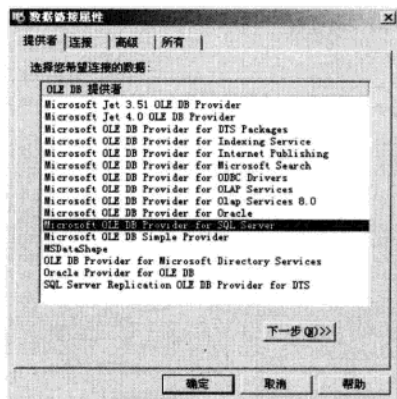


图 8.22 “数据链接属性”对话框

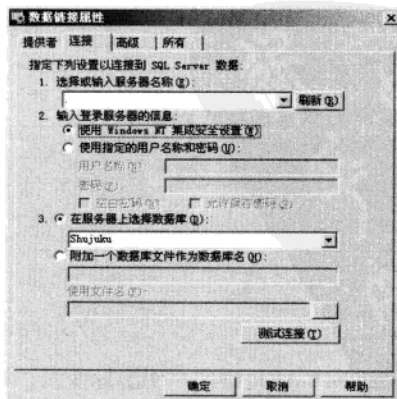


图 8.23 “连接”选项卡

(4) 单击“确定”按钮,完成对“数据链接属性”的设置。返回到图 8.21 所示的对话框中,选择 RecordSource 选项卡,在 Command Type 列表框中选择 4-adCmdStoredProc,在 Table or Stored Procedure Name 列表中选择存储过程 asdasd;1,完成设置,如图 8.24 所示。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 根据“技术要点”中的方法连接存储过程 asdasd。

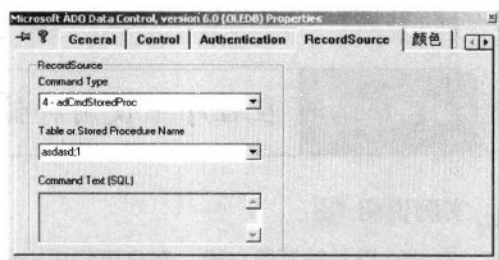


图 8.24 RecordSource 选项卡

## 举一反三

根据本实例,读者可以:

- 在应用程序中使用存储过程修改与删除数据;
- 在应用程序中使用存储过程创建数据表。

## 实例 278

## 调用具有输出参数的存储过程

这是一个可以启发思维的实例

实例位置: 光盘\mingrsoft\08\278

## 实例说明

本实例实现的是调用具有输出参数的存储过程。运行程序,在“姓名”文本框中输入“白 XX”,单击“调用”按钮,“白 XX”的手机号码将显示在“移动电话”文本框中,如图 8.25 所示。



图 8.25 调用具有输出参数的存储过程

## 技术要点

本实例的存储过程代码如下:

```
create procedure aaa
@name varchar(20), --输入的参数
@college nvarchar(50) OUTPUT --返回的参数
as
begin
select @college=移动电话
from friend
where 姓名 = @name
return
end
GO
```

使用 DECLARE 语句声明游标变量,语法如下:

```
DECLARE @local_variable data_type
```

参数说明:

- @local\_variable: 变量的名称。变量名必须以@符号开头, 局部变量名必须符合标识符规则。
- data\_type: 任何由系统提供的或用户定义的数据类型。变量不能是 text、ntext 或 image 数据类型。

使用 EXEC 执行存储过程。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加一个列表视图控件、两个编辑框控件和两个按钮控件, 为控件添加变量。
- (3) 主要程序代码如下:

```
void CReturnPataDlg::OnButtransfer()
{
 UpdateData(true);
 OnInitADOConn();
 CString str;
 str.Format("declare @c nvarchar(50) exec aaa @name='%s',@college=@c OUTPUT select @c as 移动电话",m_Name);
 //执行存储过程
 m_pRecordset->Open((_bstr_t)str,m_pConnection.GetInterfacePtr(),adOpenDynamic,adLockOptimistic,adCmdText);
 if(!m_pRecordset->adoEOF)
 {
 m_Edit = (char*)(_bstr_t)m_pRecordset->GetCollect("移动电话");
 }
 ExitConnect();
 UpdateData(false);
}
```

## 举一反三

根据本实例, 读者可以:

- 开发具有多个输出参数的存储过程。

## 实例 279 编写扩展存储过程

这是一个可以启发思维的实例

实例位置: 光盘\mingrisoft\08\279

## 实例说明

本实例实现的是调用具有输出参数的存储过程。运行程序, 在“姓名”文本框中输入“白 XX”, 单击“调用”按钮, “白 XX”的移动电话号码将显示在“移动电话”文本框中, 如图 8.26 所示。

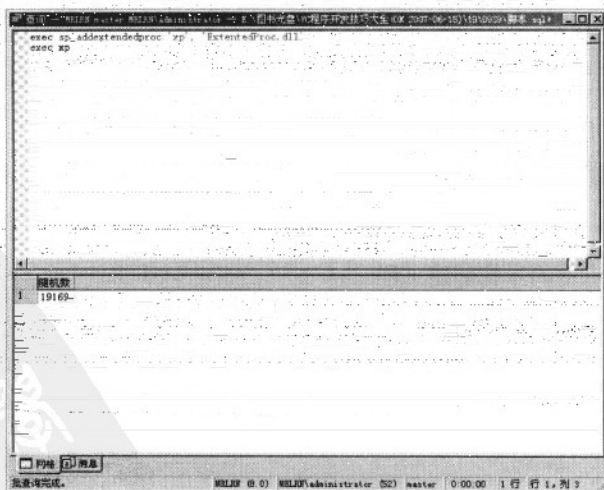


图 8.26 调用具有输出参数的存储过程



## 技术要点

利用 Visual C++ 编写扩展存储过程非常简单, 具体步骤如下。

(1) 利用扩展储存过程向导创建一个工程, 如图 8.27 所示。

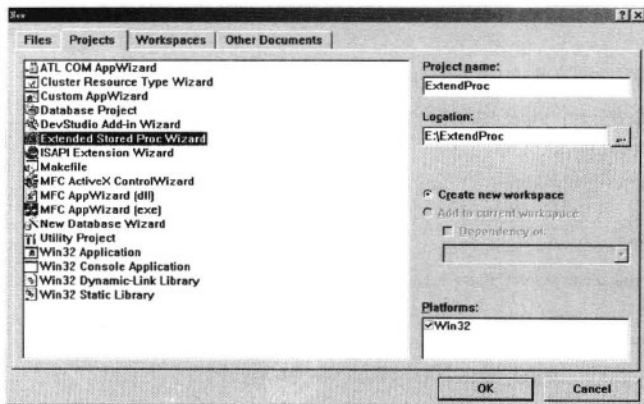


图 8.27 工程向导

(2) 单击“OK”按钮进入扩展存储过程向导, 如图 8.28 所示。

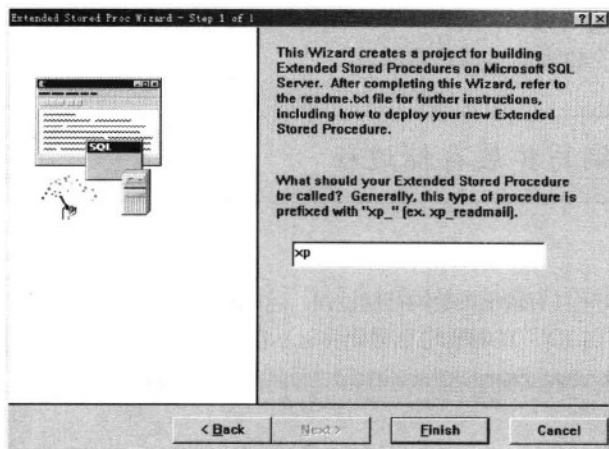


图 8.28 扩展存储过程向导

(3) 在编辑框中输入扩展存储过程的名称, 本例为“xp”。单击“Finish”按钮完成工程的创建。

## 实现过程

(1) 按照技术要点中的步骤创建工程。

(2) 主要程序代码如下:

```
RETCODE __declspec(dllexport) xp(SRV_PROC *srvproc)
{
 DBSMALLINT i = 0;
 DBCHAR colname[MAXCOLNAME];
 DBCHAR spName[MAXNAME];
 DBCHAR spText[MAXTEXT];
 wsprintf(spName, "xp");
 wsprintf(spText, "%s Sample Extended Stored Procedure", spName);
 srv_sendmsg(
```

```

 srvproc,
 SRV_MSG_INFO,
 0,
 (DBTINYINT)0,
 (DBTINYINT)0,
 NULL,
 0,
 0,
 spText,
 SRV_NULLTERM);

wprintf(colname, "随机数"); //输出标题
srv_describe(srvproc, 1, colname, SRV_NULLTERM, SRVINT2, sizeof(DBSMALLINT), SRVINT2,
 sizeof(DBSMALLINT), 0);
int num = rand(); //获得随机数
srv_setcoldata(srvproc, 1, &num);
srv_sendrow(srvproc);
srv_senddone(srvproc, SRV_DONE_MORE | SRV_DONE_COUNT, (DBSMALLINT)0, (DBINT)i);
return num; //XP_NOERROR;
}

```

(3) 运行程序, 将生成的 DLL 文件复制到 SQL Server 的安装目录 80\Tools\Binn 下, 启动 SQL Server 服务器, 在查询分析器中导入扩展存储过程。

### 举一反三

根据本实例, 读者可以:

- 开发具有多个输出参数的存储过程。

## 8.7 数据库结构的读取与修改

随着数据库的频繁使用, 数据库中的表逐渐增多, 如果需要获得表中的字段信息常常要把每个表都打开查看, 这就给工作带来了极大的不便, 所以在不开表的情况下获得数据库的结构信息变得非常重要, 本节将介绍如何通过程序获得数据库的结构信息。

### 实例 280 读取 Access 数据库结构

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\08\280

#### 实例说明

本实例实现的是读取 Access 数据库结构的功能。运行程序, 单击“打开数据库”按钮, 选择 Access 数据库文件, 单击“打开”按钮后, 数据库所在路径将显示在文本框中, 数据库中的所有数据表将显示在“表名”列表框中。在“表名”列表框中双击指定的数据表, 其表结构将显示在右侧的表格中, 结果如图 8.29 所示。

#### 技术要点

ADO 中 Connection 对象的 OpenSchema 方法可以从提供者获取数据库纲要信息, 如数据库、数据表字段类型等。下面介绍其语法及用法。

语法如下:

```
RecordsetPtr OpenSchema(enmu SchemaEnum Schema, const _variant_t &Restrictions = vtMissing, const _variant_t &SchemaID = vtMissing);
```

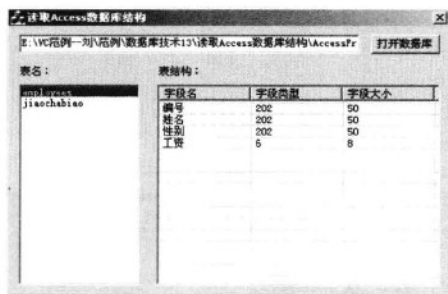


图 8.29 读取 Access 数据库结构

返回值: 返回包含纲要信息的 Recordset 对象。Recordset 将以只读、静态游标打开。

参数说明:

- Schema: 所要运行纲要的查询类型。
- Restrictions: 默认变量, 每个 Schema 选项的查询限制条件数组。
- SchemaID: 由于 OLE DB 规范没有定义提供者纲要查询的 GUID, 如果 Schema 设置为 adscmaproviderspecific, 则需要该参数, 否则不使用它。

## 实现过程

(1) 新建一个基于对话框的应用程序, 将窗体标题改为“读取 Access 数据库结构”。

(2) 向窗体中添加一个编辑框控件、一个列表框控件、一个列表视图控件和一个按钮控件, 为控件添加变量。

(3) 主要程序代码如下:

```
void CADDatabaseDlg::OnButconnect()
{
 UpdateData(true);
 m_list.ResetContent();
 OnInitADOConn(m_edit);
 _bstr_t bstrSQL;
 bstrSQL = "select*from sysobjects where xtype='U'"; //查询数据表
 CString strText;
 m_pRecordset.CreateInstance(__uuidof(Recordset)); //记录集对象实例化
 m_pRecordset->Open(bstrSQL,m_pConnection.GetInterfacePtr(),adOpenDynamic,adLockOptimistic,
 adCmdText); //打开记录集
 while(m_pRecordset->adoEOF==0)
 {
 strText=(char*)(_bstr_t)m_pRecordset->GetCollect("name"); //获得表名
 m_list.AddString(strText); //将表名插入到列表框中
 m_pRecordset->MoveNext(); //向下移动记录集指针
 }
 ExitConnect();
}

//连接数据库
void CADDatabaseDlg::OnInitADOConn(CString strsjk)
{
 ::CoInitialize(NULL);
 CString strname;
 strname.Format("Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;\
 Initial Catalog=%s;Data Source=.",strsjk);
 try
 {
 m_pConnection.CreateInstance("ADODB.Connection");
 _bstr_t strConnect=strname;
 m_pConnection->Open(strConnect,"","",adModeUnknown);
 }
 catch(_com_error e)
 {
 AfxMessageBox(e.Description());
 }
}

//断开数据库连接
void CADDatabaseDlg::ExitConnect()
{
 if(m_pRecordset!=NULL)
 m_pRecordset->Close();
 m_pConnection->Close();
}
```

## 举一反三

根据本实例, 读者可以:

- 修改 Access 数据表结构;
- 获取表中所有字段及相关属性信息。

## 实例 281

## 读取 SQL Server 数据库结构

本实例可以提高工作效率

实例位置：光盘\mingrisoft\08\281

## 实例说明

本实例实现读取 SQL Server 数据库结构的功能。运行程序，在文本框中输入 SQL Server 数据库名，此数据库中所有的数据表将显示在“数据库中的表”列表中。在“数据库中的表”列表中双击要选择的数据库表，右侧窗体上将显示此数据库表的表结构，如图 8.30 所示。

## 技术要点

在 SQL Server 数据库中的系统表 sysobjects 中保存着该数据库的表和视图等信息，其中字段值是 U 的记录就是用户所创建的表。获取用户表的 SQL 语句如下：

```
SELECT * FROM sysobjects WHERE xtype='U'
```

ADO 的 Field 对象可以获得数据库结构。Field 对象的集合、方法和属性如下：

Name 属性可返回字段名。

Value 属性可查看或更改字段中的数据。

Type、Precision 和 NumericScale 属性可返回字段的基本特性。

DefinedSize 属性可返回已声明的字段大小。

ActualSize 属性可返回给定字段中数据的实际大小。

Attributes 属性和 Properties 集合可决定对于给定字段哪些类型的功能受到支持。

AppendChunk 和 GetChunk 方法可处理包含长二进制或长字符数据的字段值。

如果提供者支持批更新，可使用 OriginalValue 和 UnderlyingValue 属性在批更新期间解决字段值之间的差异。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个编辑框控件、一个列表框控件、一个列表视图控件和一个按钮控件，为控件添加变量。

- (3) 主要程序代码如下：

```
//获得数据库中的表
void CSQLFrameDlg::OnButconn()
{
 m_list.ResetContent();
 m_edit.GetWindowText(str);
 OnInitADOConn(str);
 _bstr_t bstrSQL;
 bstrSQL = "select * from sysobjects where xtype='U'"; //查询数据表
 CString strText;
 m_pRecordset.CreateInstance(__uuidof(Recordset)); //记录集对象实例化
 m_pRecordset->Open(bstrSQL, m_pConnection.GetInterfacePtr(), adOpenDynamic,
 adLockOptimistic, adCmdText); //打开记录集
 while(m_pRecordset->adoEOF==0)
 {
 strText=(char*)(_bstr_t)m_pRecordset->GetCollect("name"); //获得数据表
 m_list.AddString(strText); //插入到列表框中
 m_pRecordset->MoveNext();
 }
}
```

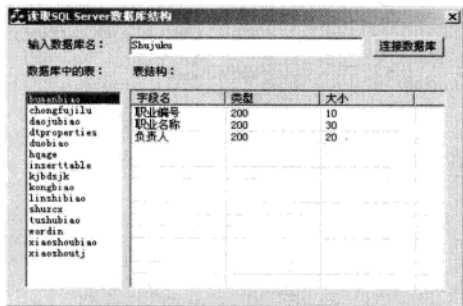


图 8.30 读取 SQL Server 数据库结构



```

 ExitConnect();
 }
 //获得数据库结构
 void CSQLFrameDlg::OnDbClickList1()
 {
 m_grid.DeleteAllItems();
 OnInitADOConn(str);
 CString str1,strname1;
 m_list.GetText(m_list.GetCurSel(),str1);
 strname1.Format("select* from %s",str1);
 _bstr_t bstrSQL;
 bstrSQL = strname1;
 m_pRecordset->Open(bstrSQL,m_pConnection.GetInterfacePtr(),adOpenDynamic,
 adLockOptimistic,adCmdText); //打开记录集
 Fields* fields=NULL;
 long count1,size1;
 BSTR bstr;
 enum DataTypeEnum stype;
 m_pRecordset->get_Fields(&fields); //获得字段结构
 count1 = fields->Count;
 for(long i=count1-1;i>=0;i--)
 {
 fields->Item[i]->get_Name(&bstr); //获得字段名
 fields->Item[i]->get_Type(&stype); //获得字段类型
 fields->Item[i]->get_DefinedSize(&size1); //获得字段大小
 //将字段信息插入到列表中
 m_grid.InsertItem(0,0);
 m_grid.SetItemText(0,0,(CString)bstr);
 m_grid.SetItemText(0,1,(char*)(_bstr_t)(long)stype);
 m_grid.SetItemText(0,2,(char*)(_bstr_t)(size1));
 }
 fields->Release();
 ExitConnect();
 }
}

```

### 举一反三

根据本实例，读者可以：

- 使用 SQL 语句建立新表；
- 删除 SQL Server 数据库中的表。

## 8.8 图片、多媒体数据录入技术

菜单是程序开发中经常使用的界面元素，合理利用菜单不但可以使用户非常方便地操作程序的功能，而且能提高效率，适应人性化的潮流。下面通过几个应用实例，介绍菜单设计的方法和技术。

### 实例 282 对 Access 数据库进行录入和提取图片

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\08\282

#### 实例说明

在一些系统中，图片数据的存取是必不可少的，如员工的照片等。许多程序设计人员将图片的路径存储在数据库中，在读取图片信息时，根据路径加载图片。这样做虽然简单，但是存在许多弊端。例如，如果磁盘中的图片丢失，就会显示错误信息。如果将图片信息存储在数据库中，图片就安全多了。本实例利用 Access 数据库存储图片信息，运行程序，在文本框中添加相关信息，单击“浏览”按钮打开图片，单击“保存”按钮将图片信息保存到数据库中，如图 8.31 所示。



图 8.31 对 Access 数据库进行录入和提取图片

## 技术要点

如果要向 Access 数据库中存储图片, 首先字段的数据类型应设置为“OLE 对象”, 先将图片以二进制的形式读入内存, 然后调用 Recordset 对象的 GetFields 方法向数据库添加图片, 将数据写入数据库中。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加 1 个图片控件、4 个编辑框控件和 4 个按钮控件, 为控件添加变量。
- (3) 保存图片, 代码如下:

```
void CBmpAccessDlg::OnBtsSave()
{
 UpdateData(true);
 if ((m_id.IsEmpty() || m_name.IsEmpty() || m_sex.IsEmpty() || m_knowledge.IsEmpty()))
 {
 MessageBox("基础信息不能为空.", "提示");
 return;
 }
 HBITMAP m_hbitmap;
 m_hbitmap = m_picture.GetBitmap();
 if (m_hbitmap == NULL)
 {
 MessageBox("请从磁盘中加载图片信息.", "提示");
 return;
 }
 m_pRecordset.CreateInstance(__uuidof(Recordset));
 m_pRecordset->Open("select * from picture", m_pConnection.GetInterfacePtr(), adOpenDynamic,
 adLockOptimistic, adCmdText);
 try
 {
 m_pRecordset->AddNew(); //添加新行
 VARIANT m_bitdata;
 CFile m_file(strText, CFile::modeRead); //打开文件
 DWORD m_filelen = m_file.GetLength()+1; //获得文件大小
 char * m_bitbuffer = new char[m_filelen];
 m_file.ReadHuge(m_bitbuffer, m_filelen); //读取文件数据
 m_file.Close(); //关闭文件
 m_bitdata.vt = VT_ARRAY|VT_UI1;
 SAFEARRAY * m_psafe;
 SAFEARRAYBOUND m_band;
 m_band.cElements = m_filelen;
 m_band.lLbound = 0;
 m_psafe = SafeArrayCreate(VT_UI1, 1, &m_band);
 for(long i=0; i < m_filelen; i++)
 {
 SafeArrayPutElement(m_psafe, &i, m_bitbuffer++);
 }
 m_bitdata.parray = m_psafe;
 //设置数据
 m_pRecordset->GetFields()->GetItem("编号")->Value = (_bstr_t)m_id;
 m_pRecordset->GetFields()->GetItem("姓名")->Value = (_bstr_t)m_name;
 m_pRecordset->GetFields()->GetItem("性别")->Value = (_bstr_t)m_sex;
 m_pRecordset->GetFields()->GetItem("学历")->Value = (_bstr_t)m_knowledge;
 m_pRecordset->GetFields()->GetItem("照片")->AppendChunk(&m_bitdata);
 m_pRecordset->Update();
 }
 catch(...)
 {
 MessageBox("操作失败");
 return;
 }
 MessageBox("操作成功.");
 m_grid.DeleteAllItems();
 AddToGrid();
}
```

- (4) 提取图片, 代码如下:

```
void CBmpAccessDlg::OnClickList1(NMHDR* pNMHDR, LRESULT* pResult)
{
 int pos;
```

```

pos=m_grid.GetSelectionMark();
m_id=m_grid.GetItemText(pos,0);
CString sql;
sql.Format("select*from picture where 编号=%s",m_id);
m_pRecordset.CreateInstance(__uuidof(Recordset));
m_pRecordset->Open((_bstr_t)sql,m_pConnection.GetInterfacePtr(),adOpenDynamic,
 adLockOptimistic,adCmdText);
HBITMAP m_hBitmap;
//读取图像字段的实际大小
long lDataSize = m_pRecordset->GetFields()->GetItem("照片")->ActualSize;
char *m_pBuffer; //定义缓冲变量
if(lDataSize > 0)
{
 //从图像字段中读取数据到varBLOB中
 _variant_t varBLOB;
 varBLOB = m_pRecordset->GetFields()->GetItem("照片")->GetChunk(lDataSize);
 if(varBLOB.vt == (VT_ARRAY | VT_UI1))
 {
 if(m_pBuffer = new char[lDataSize+1]) //分配必要的存储空间
 {
 char *pBuf = NULL;
 SafeArrayAccessData(varBLOB.parray,(void **)&pBuf);
 memcpy(m_pBuffer,pBuf,lDataSize); //复制数据到缓冲区m_pBuffer
 SafeArrayUnaccessData(varBLOB.parray);

 //将数据转换为HBITMAP格式
 LPSTR hDIB;
 LPVOID lpDIBBits;
 BITMAPFILEHEADER bmfHeader; //用于保存BMP文件头信息,包括类型、大小、位移量等
 DWORD bmfHeaderLen; //保存文件头的长度
 bmfHeaderLen = sizeof(bmfHeader); //读取文件头的长度
 //将m_pBuffer中文件头复制到bmfHeader中
 strcpy((LPSTR)&bmfHeader,(LPSTR)m_pBuffer,bmfHeaderLen);
 if (bmfHeader.bfType != (WORD*)"BM") //如果文件类型不对,则返回
 {
 MessageBox("BMP文件格式不准确");
 return;
 }
 hDIB = m_pBuffer + bmfHeaderLen; //将指针移至文件头后面
 //读取BMP文件的图像数据(包括坐标及颜色格式等信息)到BITMAPINFOHEADER对象
 BITMAPINFOHEADER &bmiHeader = *(LPBITMAPINFOHEADER)hDIB;
 //读取BMP文件的图像数据(包括坐标及颜色格式等信息)到BITMAPINFO对象
 BITMAPINFO &bmiInfo = *(LPBITMAPINFO)hDIB;
 //根据bmiHeader属性将指针移至文件头后
 lpDIBBits = (m_pBuffer) + ((BITMAPFILEHEADER *)m_pBuffer)->bOffBits;
 CClientDC dc(this); //生成一个与当前窗口相关的CClientDC,用于管理输出设置
 //生成DIBitmap数据
 m_hBitmap = CreateDIBitmap(dc.m_hDC,&bmiHeader,CBM_INIT,lpDIBBits,&bmiInfo,
 DIB_RGB_COLORS);
 }
 }
}
if(m_hBitmap != NULL)
{
 CDC* pDC = m_pictureshow.GetDC();
 CRect r;
 m_pictureshow.GetClientRect(&r);
 //将位图选进设备场景中
 CDC memdc;
 memdc.CreateCompatibleDC(pDC);
 memdc.SelectObject(m_hBitmap);
 BITMAP bmp;
 GetObject(m_hBitmap,sizeof(bmp),&bmp);
 //绘制图片
 pDC->StretchBlt(r.left,r.top,r.Width(),r.Height(),&memdc,0,0,bmp.bmWidth,bmp.bmHeight,SRCCOPY);
 memdc.DeleteDC();
}
*pResult = 0;
}
}

```

### 举一反三

根据本实例,读者可以:

- 向 SQL Server 数据库中存储和提取图片。

## 实例 283

对 SQLServer 数据库进行  
录入和提取多媒体文件

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\08\283

## 实例说明

随着数据库技术的不断发展，如今的数据库管理系统不仅能够存储数据，还能够存储图片、语音等多媒体信息。本实例演示了如何将 mp3 文件存储在 SQL Server 数据库中，效果如图 8.32 所示。

## 技术要点

在将多媒体文件存入数据库时，就是将文件写成二进制形式保存到数据库中，这些二进制数据可能会很大，所以 SQL Server 提供了一种机制，能存储每行大到 2GB 的二进制对象 (BLOB)，这类对象可包括 IMAGE、TEXT 和 NTEXT 3 种数据类型。IMAGE 数据类型存储的是二进制数据，最大长度是  $2^{31}-1$  (2147483647)，本实例使用的是 IMAGE 数据类型。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加 2 个编辑框控件、1 个组合框控件和 3 个按钮控件，为控件添加变量。
- (3) 添加多媒体文件，代码如下：

```
void CInputvoiceDlg::OnButsave()
{
 UpdateData(true);
 try
 {
 char *m_pBuffer;
 CFile file;
 if(!file.Open(strText, CFile::modeRead))
 {
 MessageBox("无法打开文件");
 return;
 }
 DWORD m_filelen;
 m_filelen = file.GetLength(); //获得文件大小
 m_pBuffer = new char[m_filelen + 1];
 if(file.ReadHuge(m_pBuffer, m_filelen) != m_filelen)
 {
 MessageBox("读取文件出现错误");
 return;
 }
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn(); //连接数据库
 _bstr_t sql;
 sql = "select*from voice";
 _RecordsetPtr m_pRecordset;
 m_pRecordset = m_AdoConn.GetRecordSet(sql);
 m_pRecordset->AddNew(); //添加新行
 VARIANT varblob;
 SAFEARRAY *psa;
 SAFEARRAYBOUND rgsabound[1];
 rgsabound[0].lLbound = 0;
 rgsabound[0].cElements = m_filelen;
 psa = SafeArrayCreate(VT_UI1, 1, rgsabound);
 for(long i=0; i<(long)m_filelen; i++)
 {
 SafeArrayPutElement(psa, &i, m_pBuffer++);
 }
 varblob.vt = VT_ARRAY|VT_UI1;
```

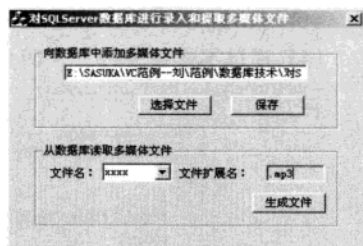


图 8.32 对 SQLServer 数据库进行  
录入和提取多媒体文件



```

varblob.parray = psa;
//添加数据
m_pRecordset->GetFields()->GetItem("name")->Value = (_bstr_t)strName;
m_pRecordset->GetFields()->GetItem("postfix")->Value = (_bstr_t)strkzm;
m_pRecordset->GetFields()->GetItem("voice")->AppendChunk(varblob);
m_pRecordset->Update();
m_AdoConn.ExitConnect();
}
catch(...)
{
 MessageBox("操作失败");
 return;
}
MessageBox("操作成功.");
m_combo.ResetContent();
AddToCombo();
}

```

#### (4) 提取多媒体文件, 代码如下:

```

void CInputvoiceDlg::OnButcreate()
{
 CString name,kzm;
 m_combo.GetLBText(m_combo.GetCurSel(),name);
 if(name=="")
 {
 MessageBox("请选择文件名!");
 return;
 }
 m_kzm.GetWindowText(kzm);
 try
 {
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn();
 //设置SELECT语句
 _bstr_t vSQL;
 vSQL = "select*from voice where name='"+name+"' ";
 //执行SELETE语句
 _RecordsetPtr m_pRecordset;
 m_pRecordset = m_AdoConn.GetRecordSet(vSQL);
 long IDataSize = m_pRecordset->GetFields()->GetItem("voice")->ActualSize;
 char *m_pBuffer; //定义缓冲变量
 if(IDataSize > 0)
 {
 //从图像字段中读取数据到varBLOB中
 _variant_t varBLOB;
 varBLOB = m_pRecordset->GetFields()->GetItem("voice")->GetChunk(IDataSize);
 if(varBLOB.vt == (VT_ARRAY | VT_UI1))
 {
 if(m_pBuffer = new char[IDataSize+1]) //分配必要的存储空间
 {
 char *pBuf = NULL;
 SafeArrayAccessData(varBLOB.parray,(void **)&pBuf);
 memcpy(m_pBuffer,pBuf,IDataSize); //复制数据到缓冲区m_pBuffer
 SafeArrayUnaccessData (varBLOB.parray);
 }
 }
 char buff[256];
 ::GetCurrentDirectory(256,buff); //获得程序根目录
 strcat(buff,"\\");
 strcat(buff,name);
 strcat(buff,kzm);
 CFile file _T(buff),CFile::modeCreate|CFile::modeWrite); //创建文件
 file.Write(m_pBuffer,IDataSize); //向文件中写入数据
 file.Flush();
 file.Close();
 }
 }
 catch(...)
 {
 MessageBox("操作失败");
 return;
 }
 MessageBox("操作成功.");
}

```

## 举一反三

根据本实例，读者可以：

- 向数据库中存储音频、视频数据。

## 8.9 数据备份恢复

计算机系统在运行的过程中，可能由于硬件故障、系统软件和应用软件的错误、遭到病毒攻击、环境因素等多种原因而造成故障，因此用户应该经常对系统内的数据库进行备份。这样在故障产生后，用户可通过系统中的恢复功能将备份的数据恢复到系统中，从而减少故障给用户带来的损失。

## 实例 284 Access 数据库备份与还原

本实例可以提高数据库安全性

实例位置：光盘\mingrsoft\08\284

## 实例说明

计算机系统在运行的过程中难免会出现硬件故障、系统软件和应用软件的错误，而造成计算机瘫痪或应用软件无法运行，这样数据库的备份与恢复就显得特别重要，尤其在商务软件中，一旦数据丢失，后果不堪设想。那么如何在 Visual C++ 中实现 Access 数据库的备份与还原呢？运行程序，单击“保存文件路径”按钮，设置备份文件的保存路径，单击“备份”按钮，即可实现 Access 数据库的备份，如图 8.33 所示。如果要还原 Access 数据库，则首先选中“还原”单选按钮，单击“选择备份文件”按钮，选择要进行还原的 Access 数据库文件，单击“还原”按钮，即可实现 Access 数据库的还原。

## 技术要点

数据备份对数据库的安全来说是至关重要的。数据备份是指在某种介质（磁带、磁盘等）上存储数据库的复制。数据还原是指及时将数据库返回到原来状态。

用复制文件的原理可以实现数据备份与数据还原，所以在程序中用到了 CopyFile 语句，语法如下：

```
BOOL CopyFile(LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName, BOOL bFailIfExists);
```

参数说明：

- lpExistingFileName：源文件名。
- lpNewFileName：目标文件名。
- bFailIfExists：如果设为 TRUE（非零），那么一旦目标文件已经存在，则函数调用会失败，否则目标文件被改写。

## 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“Access 数据库备份与还原”。
- (2) 在窗体上添加一个编辑框控件、两个单选按钮控件和两个按钮控件，为控件添加变量。
- (3) 主要程序代码如下：

```
void CBackUpAccessDlg::OnButbackup()
```

```
{
 CopyFile(m_edit1,m_edit2,false); //复制文件
 if (radio)
 {
```

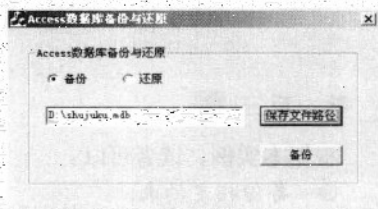


图 8.33 Access 数据库备份与恢复

```
 MessageBox("备份完成!", "系统提示", MB_OK|MB_ICONEXCLAMATION);
 }
 else
 {
 MessageBox("还原完成!", "系统提示", MB_OK|MB_ICONEXCLAMATION);
 }
}
//获得文件保存路径
void CBackUpAccessDlg::OnBtSave()
{
 //创建文件浏览对话框
 CString ReturnPach;
 TCHAR szPath[_MAX_PATH];
 BROWSEINFO bi;
 bi.hwndOwner=NULL;
 bi.pidlRoot=NULL;
 if(radio)
 {
 bi.lpszTitle=_T("请选择备份文件夹");
 }
 else
 {
 bi.lpszTitle=_T("请选择还原文件夹");
 }
 bi.pszDisplayName=szPath;
 bi.ulFlags=BIF_RETURNONLYFSDIRS;
 bi.lpfnc=NULL;
 bi.lParam=NULL;
 LPITEMIDLIST pItemIDList=SHBrowseForFolder(&bi);
 if(pItemIDList)
 {
 if(SHGetPathFromIDList(pItemIDList,szPath))
 ReturnPach=szPath;
 }
 else
 ReturnPach="";
 m_edit2 = ReturnPach;
 m_edit2 = m_edit2 + strName;
 UpdateData(false);
}
```

### 举一反三

根据本实例，读者可以：

- 备份指定的表；
- 按用户设置备份数据库。

## 实例 285

## SQL Server 数据库备份与恢复

本实例可以提高数据库安全性

实例位置：光盘\mingrisoft\08\285

### 实例说明

本实例实现的是在 Visual C++ 中实现 SQL Server 数据库的备份与恢复。运行程序，输入备份文件名，单击“选择备份路径”按钮选择保存路径，单击“备份数据库”按钮，将对程序中指定的 SQL Server 数据库文件进行备份，如图 8.34 所示。如果要恢复 SQL Server 数据库，则单击“恢复数据库”按钮，即可实现 SQL Server 数据库的恢复。

### 技术要点

运用 Transact-SQL 中的 BACKUP 命令和 RESTORE 命令可以完成 SQL Server 数据库的备份

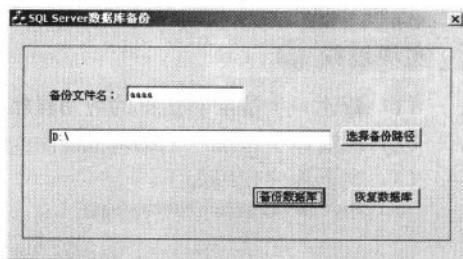


图 8.34 SQL Server 数据库备份与恢复

与恢复。BACKUP 命令用来备份 SQL Server 数据库, RESTORE 命令用来恢复使用 BACKUP 命令所做的备份。下面是这两个命令的语法及其相关用法。

(1) BACKUP 数据库备份命令。备份整个数据库、事务日志或者备份一个或多个文件或文件组。

语法如下:


```
BACKUP DATABASE { database_name | @database_name_var }
TO < backup_device > [,...n]
[WITH
 [BLOCKSIZE = { blocksize | @blocksize_variable }]
 [[,] DESCRIPTION = { 'text' | @text_variable }]
 [[,] EXPIREDATE = { date | @date_var }
 [RETAINDAYS = { days | @days_var }]
 [[,] PASSWORD = { password | @password_variable }]
 [[,] FORMAT | NOFORMAT]
]
```

BACKUP 命令的语法包含如下部分。


- DATABASE: 指定一个完整的数据库备份。假如指定了一个文件和文件组的列表, 那么仅有这些被指定的文件和文件组被备份。

{ database\_name | @database\_name\_var }: 指定一个数据库, 从该数据库中对事务日志、部分数据库或完整的数据库进行备份。如果作为变量 (@database\_name\_var) 提供, 则可将该名称指定为字符串常量 (@database\_name\_var = database name) 或字符串数据类型 (ntext 或 text 数据类型除外) 的变量。

- < backup\_device >: 指定备份操作时要使用的逻辑或物理备份设备。当指定 TO Disk 或 TO Tape 时, 应输入完整路径和文件名。

 说明: 对于备份到磁盘的情况, 如果输入一个相对路径名, 备份文件将存储到默认的备份目录中。

- n: 表示可以指定多个备份设备的占位符。备份设备数目的上限为 64。
- EXPIREDATE = { date | @date\_var }: 指定备份集到期和允许被重写的日期。如果将该日期作为变量 (@date\_var) 提供, 则可以将该日期指定为字符串常量 (@date\_var = date)、字符串数据类型变量 (ntext 或 text 数据类型除外)、smalldatetime 或者 datetime 变量, 并且该日期必须符合已配置的系统 datetime 格式。
- PASSWORD = { password | @password\_variable }: 为备份集设置密码, password 是一个字符串。如果为备份集定义了密码, 必须提供这个密码才能对该备份集执行任何还原操作。
- FORMAT: 指定应将媒体头写入用于此备份操作的所有卷。任何现有的媒体头都被重写。FORMAT 选项使整个媒体内容无效, 并且忽略任何现有的内容。

 注意: 使用 FORMAT 要谨慎。格式化一个备份设备或媒体将使整个媒体集不可用。例如, 如果初始化现有磁带备份集中的单个磁带, 则整个备份集都将变得不可用。

(2) RESTORE 数据库恢复命令。RESTORE 数据库恢复命令主要用于还原使用 BACKUP 命令所做的备份。

语法如下:

```
RESTORE DATABASE { database_name | @database_name_var }
FROM < backup_device > [,...n]
[WITH
 [RESTRICTED_USER]
 [[,] FILE = { file_number | @file_number }]
 [[,] PASSWORD = { password | @password_variable }]
 [[,] MOVE 'logical_file_name' TO 'operating_system_file_name']
 [[,] ...n]
 [[,] REPLACE]
 [[,] RESTART]
]
```



RESTORE 命令包含如下部分。

DATABASE: 指定从备份还原整个数据库。

- {database\_name | @database\_name\_var}: 将日志或整个数据库还原到的数据库。如果将其作为变量 (@database\_name\_var) 提供, 则可将该名称指定为字符串常量 (@database\_name\_var = database\_name) 或字符串数据类型 (ntext 或 text 数据类型除外) 的变量。
- FROM: 指定从中还原备份的备份设备。如果没有指定 FROM 子句, 则不会发生备份还原, 而是恢复数据库。可用省略 FROM 子句的办法尝试恢复通过 NORECOVERY 选项还原的数据库或切换到一台备用服务器上。如果省略 FROM 子句, 则必须指定 NORECOVERY、RECOVERY 或 STANDBY。
- <backup\_device>: 指定还原操作要使用的逻辑或物理备份设备。{'logical\_backup\_device\_name' | @logical\_backup\_device\_name\_var} 是由 sp\_addumpdevice 创建的备份设备 (数据库将从该备份设备还原) 的逻辑名称, 该名称必须符合标识符规则。如果作为变量 (@logical\_backup\_device\_name\_var) 提供, 则可以指定字符串常量 (@logical\_backup\_device\_name\_var = logical\_backup\_device\_name) 或字符串数据类型 (ntext 或 text 数据类型除外) 的变量作为备份设备名。
- RESTRICTED\_USER: 限制只有 db\_owner、dbcreator 或 sysadmin 角色的成员才能访问最近还原的数据库。在 SQL Server 2000 中, RESTRICTED\_USER 替换了选项 DBO\_ONLY。提供 DBO\_ONLY 只是为了向后兼容。
- FILE = { file\_number | @file\_number }: 标识要还原的备份集。例如, file\_number 为 1 表示备份媒体上的第一个备份集, file\_number 为 2 表示第二个备份集。
- PASSWORD = { password | @password\_variable }: 提供备份集的密码。PASSWORD 是一个字符串。如果在创建备份集时提供了密码, 则从备份集执行还原操作时必须提供密码。
- MOVE 'logical\_file\_name' TO 'operating\_system\_file\_name': 指定应将给定的 logical\_file\_name 移到 operating\_system\_file\_name。默认情况下, logical\_file\_name 将还原到其原始位置。如果使用 RESTORE 语句将数据库复制到相同或不同的服务器上, 则可能需要使用 MOVE 选项重新定位数据库文件以避免与现有文件冲突。可以在不同的 MOVE 语句中指定数据库内的每个逻辑文件。
- n: 占位符, 表示可通过指定多个 MOVE 语句移动多个逻辑文件。
- REPLACE: 指定即使存在另一个具有相同名称的数据库, SQL Server 也应该创建指定的数据库及其相关文件。在这种情况下将删除现有的数据库。如果没有指定 REPLACE 选项, 则将进行安全检查以防止意外重写其他数据库。当 RESTORE 语句中命名的数据库已经在当前服务器上存在, 并且该数据库名称与备份集中记录的数据库名称不同时, 进行安全检查, RESTORE DATABASE 语句不会将数据库还原到当前服务器。

RESTART: 指定 SQL Server 应重新启动被中断的还原操作。RESTART 从中断点重新启动还原操作。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加 2 个编辑框控件和 3 个按钮控件, 为控件添加变量。
- (3) 主要程序代码如下:

```
//备份数据库
void CSQLstockupDlg::OnOK()
{
 UpdateData(true);
```

```

ADOConn m_AdoConn;
CString sql,str,path;
m_edit.GetWindowText(str);
strpath = str+"\\\\"+m_name+".dat";
sql = "use master exec sp_addumpdevice 'disk','"+m_name+"','"+strpath+" backup database\
shujuku to '"+m_name+" '"; //设置备份语句
m_AdoConn.ExecuteSQL((_bstr_t)sql); //备份数据库
m_AdoConn.ExitConnect();
MessageBox("备份完成!", "系统提示", MB_OK|MB_ICONEXCLAMATION);
//CDialog::OnOK();
}
//恢复数据库
void CSQLstockupDlg::OnButhf()
{
 UpdateData(true);
 ADOConn m_AdoConn;
 CString sql;
 sql = "use master restore database shujuku from '"+m_name+" '"; //设置恢复语句
 m_AdoConn.ExecuteSQL((_bstr_t)sql); //恢复数据库
 m_AdoConn.ExitConnect();
 MessageBox("恢复完成!", "系统提示", MB_OK|MB_ICONEXCLAMATION);
}

```

## 举一反三

根据本实例，读者可以：

- 定期备份 SQL Server 数据库。
- 备份指定的数据表。

## 实例 286 定时数据备份

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\08\286

## 实例说明

定时备份数据库可以有效地防止数据丢失。运行程序，设置每天备份数据库的时间、备份的数据库和备份路径，如图 8.35 所示，当系统时间与设置的备份时间吻合时，则自动备份数据库到指定的路径下，如图 8.36 所示。

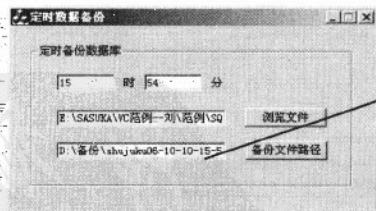


图 8.35 设置备份信息

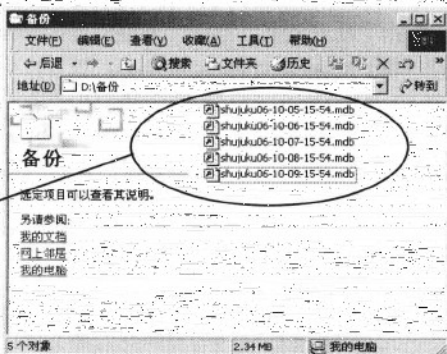


图 8.36 备份后的数据库

## 技术要点

定时备份数据库是通过设置定时器来实现的，程序每分钟进行一次判断，看系统时间是否和设置的时间相同，如果相同则备份数据库。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加 4 个编辑框控件和 2 个按钮控件, 为控件添加变量。
- (3) 主要程序代码如下:

```
void CTimeBackupDlg::OnTimer(UINT nIDEvent)
{
 CTime time;
 //获得当前系统时间
 time=time.GetCurrentTime();
 hour = time.GetHour();
 min = time.GetMinute();
 //获得控件中设置的时间
 CString mhour,mmin;
 m_hour.GetWindowText(mhour);
 m_min.GetWindowText(mmin);
 if(hour == atoi(mhour) && min == atoi(mmin)) //比较时间
 {
 CopyFile(m_edit1,m_edit2,false); //相同时复制数据库
 }
 CDialog::OnTimer(nIDEvent);
}
```

## 举一反三

根据本实例, 读者可以:

- 在指定的时间间隔内备份数据库。

## 8.10 其他数据库技术

## 实例 287

断开 SQL Server 数据库  
与其他应用程序的连接

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\08\287

## 实例说明

要断开 SQL Server 数据库与其他应用程序的连接, 需要停止 SQL Server 服务管理器。运行程序, 单击“断开连接”按钮, 如图 8.37 所示, 打开 SQL Server 服务管理器, 会发现 SQL Server 服务管理器已停止, 如图 8.38 所示。

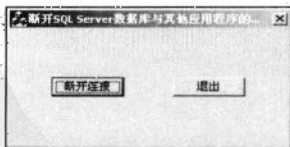


图 8.37 断开 SQL Server 数据库连接

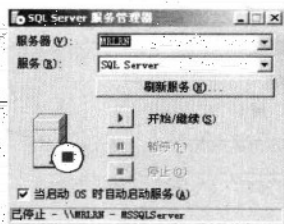


图 8.38 SQL Server 服务管理器

## 技术要点

在 SQL Server 数据库中, 可从本地服务器、远程客户端或另一台服务器停止 SQL Server 实例。如果没有暂停 SQL Server 实例便停止服务器, 所有服务器进程将立即终止。停止 SQL

Server 实例可防止新连接并与当前用户断开连接。

SHUTDOWN 语句在 osql 或其他查询工具中执行时停止 SQL Server 实例。使用 WITH NOWAIT 选项可立即停止 SQL Server 实例。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加两个按钮控件。
- (3) 主要程序代码如下：

```
void CCloseConnectDlg::OnButclose()
{
 OnInitADOConn();
 _bstr_t sql;
 sql = "SHUTDOWN WITH NOWAIT";
 ExecuteSQL(sql); //断开数据库连接
 ExitConnect();
}
```

## 举一反三

根据本实例，读者可以：

- 远程停止 SQL Server 服务管理器。

## 实例 288 在 Visual C++ 中执行事务

这是一个可以提高基础技能的实例

实例位置：光盘\mingrsoft\08\288

## 实例说明

事务是指作为单个工作单元执行的一系列操作。本实例实现的是当修改数据信息后单击“修改”按钮时，弹出是否确认保存修改信息的提示信息，如图 8.39 所示。单击“是”按钮保存修改的信息，单击“否”按钮则不保存修改的信息。

## 技术要点

通过事务可以实现是否确认修改信息的功能。在进行修改信息的时候，执行 Connection 对象的 BeginTrans 方法执行开始事务，随后弹出是否确认保存的提示对话框，如果保存修改则执行 Connection 对象的 CommitTrans 提交事务；如果不保存修改则执行 Connection 对象的 RollbackTrans 方法回滚事务，保持原来的信息不变。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加 4 个编辑框控件、1 个按钮控件、1 个 ADO Data 控件和 1 个 DataGrid 控件，为控件添加变量。
- (3) 主要程序代码如下：

```
void CAffairDlg::OnButtonmod()
{
 UpdateData(true);
 OnInitADOConn();
 m_pConnection->BeginTrans();
```

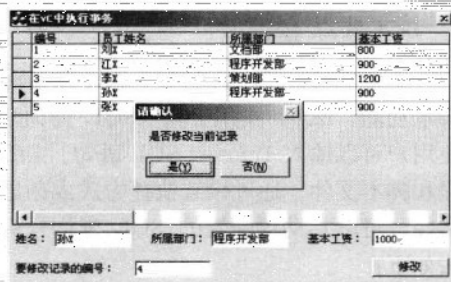


图 8.39 在 VC 中执行事务



```
CString bstrSQL;
bstrSQL.Format("update tb_laborage set 员工姓名='%s',所属部门='%s',基本工资=%d where \
 编号=%d",m_Name,m_Dep,atoi(m_Laborage),atoi(m_Id)); //设置修改语句
m_pConnection->Execute((_bstr_t)bstrSQL,NULL,adCmdText);
if(MessageBox("是否修改当前记录","请确认",MB_YESNO)==IDYES)
{
 m_pConnection->CommitTrans(); //提交事务
}
else
{
 m_pConnection->RollbackTrans(); //回滚事务
}
ExitConnect();
m_Adodc.SetRecordSource("select * from tb_laborage");
m_Adodc.Refresh();
}
```

### 举一反三

根据本实例，读者可以：

- 在取消数据修改时执行数据回滚事务。

## 实例 289 在程序中执行 SQL 脚本

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\08\289

### 实例说明

本实例实现的是在程序中执行 SQL 脚本，并将运行结果保存到文本文件中。运行程序，输入服务器和数据库，单击“选择 SQL 脚本”按钮选择一个 SQL 脚本，单击“执行”按钮运行 SQL 脚本，如图 8.40 所示。

### 技术要点

本实例通过 isqlw 实用工具来实现执行 SQL 脚本功能。isqlw 实用工具（SQL 查询分析器）使用户可以输入 Transact-SQL 语句、系统存储过程 and 脚本文件。通过设置快捷方式或创建批处理文件，可以启动预配置的 SQL 查询分析器，语法如下：

```
isqlw
[?]
[
 [-S server_name[instance_name]]
 [-d database]
 [-E] [-U user] [-P password]
 [{-i input_file} {-o output_file} [-F {U|A|O}]]
 [-f file_list]
 [-C configuration_file]
 [-D scripts_directory]
 [-T template_directory]
]
```

参数说明：

- -?: 显示用法信息。
- -S server\_name[instance\_name]: 指定要连接到的 SQL Server 服务器。
- -d database: 当启动 isqlw 时，发出一个 USE database 语句。默认值为用户的默认数据库。

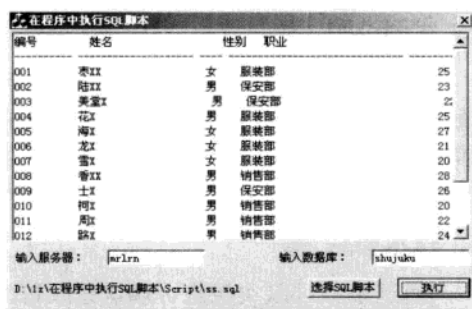


图 8.40 在程序中执行 SQL 脚本

- -E: 使用信任连接而不请求密码。
- -U user: 用户登录 ID。登录 ID 区分大小写。
- -P password: 登录密码, 默认设置为 NULL。
- -i input\_file: 标识包含一批 SQL 语句或存储过程的文件, 必须同时指定 -i 和 -o 选项。如果指定 -i 和 -o 选项, 将执行输入文件中的查询, 并将结果保存到输出文件中。在查询执行过程中不显示用户接口。当执行完成后, 进程退出。
- -o output\_file: 标识接收来自 isqlw 的输出文件, 必须同时指定 -i 和 -o 选项。如果指定 -i 和 -o 选项, 将执行输入文件中的查询, 并将结果保存到输出文件中。在查询执行过程中不显示用户接口。当执行完成后, 进程退出。如果未使用 -F 指定文件格式, 则输出文件使用与输入文件相同的类型。
- -F {U|A|O}: 输入文件和输出文件的格式, 值包括 Unicode、ANSI 和 OEM。如果未指定 -F, 则使用自动模式 (如果文件标为 Unicode 格式, 则以 Unicode 格式打开; 否则, 以 ANSI 格式打开文件)。
- -f file\_list: 将列出的文件装载到 SQL 查询分析器中。使用 -f 选项, 可以装载一个或多个文件 (文件名以单个空格分开)。如果指定了多个文件, 则以相同的连接上下文将这些文件打开。文件名可以包含该文件所驻留的目录路径。
- -C configuration\_file: 使用配置文件中指定的设置。其他在命令提示下显式指定的参数将重写相应配置文件中的设置。
- -D scripts\_directory: 重写在注册表中或在用 -C 指定的配置文件中指定的默认存储脚本目录。该值不保留在注册表或配置文件中。若要在 SQL 查询分析器中查看该选项的当前值, 单击“工具”菜单按钮, 然后选择“选项”命令。
- -T template\_directory: 重写在注册表中或在用 -C 指定的配置文件中指定的默认模板目录, 该值不保留在注册表或配置文件中。若要在 SQL 查询分析器中查看该选项的当前值, 单击“工具”菜单按钮, 然后选择“选项”命令。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加一个 RichEdit 控件、两个编辑控件和两个按钮控件, 为控件添加变量。
- (3) 主要程序代码如下:

```
void CScriptDlg::OnButexecute()
{
 UpdateData(true);
 char buf[256];
 ::GetCurrentDirectory(256, buf); // 获取程序根目录路径
 strext(buf, "\\sql.txt");
 CString StrPath, sqltxt;
 sqltxt = buf;
 StrPath.Format("isqlw -S %s -d %s -E -i %s -o %s", m_Server, m_Database, strText, sqltxt); // 设置执行命令
 WinExec(StrPath, 4); // 执行 SQL 脚本
 Sleep(1000);
 CString str = "";
 char sread[10000];
 CFile mfile(_T(sqltxt), CFile::modeRead); // 打开文件
 mfile.Read(sread, 10000); // 读取文件内容
 for(int i=0; i<mfile.GetLength(); i++)
 {
 str += sread[i];
 }
 m_RichEdit.SetWindowText(str);
}
```

## 举一反三

根据本实例，读者可以：

- 在程序中执行存储过程。

## 实例 290 利用 SQL 语句执行外围命令

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\08\290

## 实例说明

本实例实现的是利用 SQL 语句执行外围命令。运行程序，选择一个文本文件，并设置其保存路径，单击“复制”按钮将文本文件复制到设置的路径下，如图 8.41 所示。

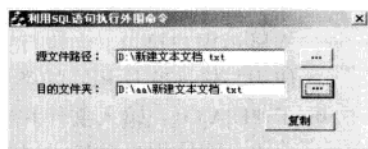


图 8.41 利用 SQL 语句执行外围命令

## 技术要点

本实例使用 xp\_cmdshell 扩展过程来实现文件的复制，xp\_cmdshell 扩展过程以操作系统命令行解释器的方式执行给定的命令字符串，并以文本行方式返回任何输出。授予非管理用户执行 xp\_cmdshell 的权限，语法如下：

```
xp_cmdshell {'command_string'} [, no_output]
```

参数说明：

- command\_string：在操作系统命令行解释器上执行的命令字符串。command\_string 的数据类型为 varchar(255)或 nvarchar(4000)，没有默认值。command\_string 不能包含一对以上的双引号。如果由 command\_string 引用的文件路径或程序名称中有空格，则需要使用一对引号。
- no\_output：是可选参数，表示执行给定的 command\_string，但不向客户端返回任何输出。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加 2 个编辑框控件和 3 个按钮控件，为控件添加变量。
- (3) 主要程序代码如下：

```
void CPeripheryCommandDlg::OnButcopy()
{
 UpdateData(true);
 OnInitADOConn();
 CString bstrSQL;
 bstrSQL.Format("xp_cmdshell 'copy %s %s', NO_OUTPUT", m_Edit1, m_Edit2); //设置执行命令
 m_pConnection->Execute((_bstr_t)bstrSQL, NULL, adCmdText); //执行命令
 ExitConnect();
 MessageBox("复制成功！");
}
```

## 举一反三

根据本实例，读者可以：

- 获得可执行文件列表。

## 实例 291 枚举 SQL Server 服务器

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\08\291

## 实例说明

在使用一些带有数据库的程序时，有时需要修改服务器信息，本实例实现的是枚举当前所

有的 SQL Server 服务器,运行程序,所有的 SQL Server 服务器都将插入到列表中,如图 8.42 所示。

## 技术要点

要枚举 SQL Server 服务器,需要向程序中引入头文件 sql.h 和 sqltext.h,导入 odbc32.lib 动态库,然后通过 SQLAllocHandle 等方法分配环境句柄进行设置。

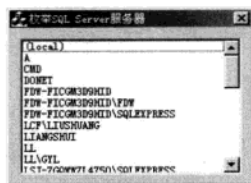


图 8.42 枚举 SQL Server 服务器

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个列表控件控件,为其添加成员变量 m\_Server。
- (3) 引入头文件 sql.h 和 sqltext.h,导入 odbc32.lib 动态库。
- (4) 添加函数 GetServer,该函数用于获得服务器字符串,代码如下:

```
CString CEnumServerDlg::GetServer()
{
 CString sSQLChar = "Driver={SQL Server}";
 CString cKey = "SERVER";
 SQLHENV hSqlHenv;
 SQLHDBC hSQLHdbc;
 short sConnStrOut;
 CString Returnstr;
 //分配环境句柄
 int IsSuccess=SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&hSqlHenv);
 if (IsSuccess == SQL_SUCCESS || IsSuccess == SQL_SUCCESS_WITH_INFO)
 {
 //设置环境属性
 IsSuccess = SQLSetEnvAttr(hSqlHenv, SQL_ATTR_ODBC_VERSION,
 (void *)SQL_OV_ODBC3, 0);
 if (IsSuccess == SQL_SUCCESS || IsSuccess == SQL_SUCCESS_WITH_INFO)
 {
 //分配一个连接句柄
 IsSuccess = SQLAllocHandle(SQL_HANDLE_DBC, hSqlHenv, &hSQLHdbc);
 if (IsSuccess == SQL_SUCCESS || IsSuccess == SQL_SUCCESS_WITH_INFO)
 {
 CString szConnStrOut;
 //调用SQLBrowseConnect
 IsSuccess = SQLBrowseConnect(hSQLHdbc, (SQLCHAR *)
 sSQLChar.GetBuffer(sSQLChar.GetLength()), SQL_NTS, (SQLCHAR *)
 (szConnStrOut.GetBuffer(4824)), 4824, &sConnStrOut); //获取所有服务器
 szConnStrOut.ReleaseBuffer();
 int nPos=szConnStrOut.Find(cKey);
 if(nPos!=-1)
 {
 nPos=nPos+cKey.GetLength();
 int nBegin=szConnStrOut.Find("{",nPos+1);
 int nEnd=szConnStrOut.Find("}",nPos+1);
 Returnstr=szConnStrOut.Mid(nBegin+1,nEnd-(nBegin+1));
 }
 }
 }
 }
 return Returnstr;
}
```

- (5) 添加 FormatString 函数,该函数用于分解各个服务器名,并将服务器名称插入到列表中,代码如下:

```
void CEnumServerDlg::FormatString(CString sText, CListBox *pListBox)
{
 int nPos=0,nOldPos=0;
 CString sMem;
 while(nPos!=-1)
 {
 nOldPos=nPos;
 nPos=sText.Find(",",nPos+1);
 if(nPos==-1)
 {
 nPos=sText.GetLength();
 if(nOldPos==0)
 {
 //查找“,”
 //获得字符串长度
 //获得服务器名
 }
 }
 }
}
```



```

 sMem=sText.Mid(nOldPos,nPos-nOldPos);
 else
 sMem=sText.Mid(nOldPos+1,nPos-nOldPos-1);
 if(nPos=sText.GetLength()) //如果拆分完服务器名
 nPos=-1; //退出循环
 if(sMem.IsEmpty())
 continue;
 pListBox->AddString(sMem); //插入到列表中
}
}

```

(6) 在对话框初始化时, 获得 SQL Server 服务器名代码如下:

```

BOOL CEnumServerDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//系统代码省略
 CString sServer;
 CString sText=GetServer(); //获得服务器字符串
 FormatString(sText,&m_Server); //将服务器名称插入到列表中
 return TRUE;
}

```

## 举一反三

根据本实例, 读者可以:

- 在取消数据修改时执行数据回滚事务。

## 实例 292 附加数据库

本实例是一个提高效率的程序

实例位置: 光盘\mingrisoft\08\292

## 实例说明

本实例实现的是通过程序附加 SQL Server 数据库。运行程序, 单击“...”按钮选择一个 SQL 数据库文件, 单击“附加”按钮完成数据库的附加, 如图 8.43 所示。

## 技术要点

本实例通过 `sp_attach_db` 存储过程来实现附加 SQL Server 数据库的功能。`sp_attach_db` 存储过程的语法如下:

```

sp_attach_db [@dbname =] 'dbname'
, [@filename1 =] 'filename_n' [,...16]

```

参数说明:

- [@dbname =] 'dbname': 要附加到服务器的数据库的名称。该名称必须是惟一的。dbname 的数据类型为 sysname, 默认值为 NULL。
- [@filename1 =] 'filename\_n' 数据库文件的物理名称, 包括路径。filename\_n 的数据类型为 nvarchar(260), 默认值为 NULL。最多可以指定 16 个文件名。参数名称以 @filename1 开始, 递增到 @filename16。文件名列表至少必须包括主文件, 主文件包含指向数据库中其他文件的系统表。该列表还必须包括数据库分离后所有被移动的文件。

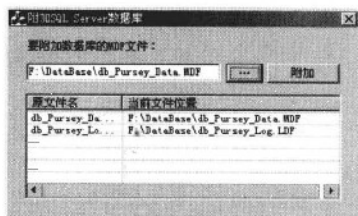


图 8.43 附件数据库

## 实现过程

- 新建一个基于对话框的应用程序。
- 向窗体中添加一个静态文本控件、一个编辑框控件、一个列表视图控件和两个按钮控

件。为编辑框控件和列表视图控件添加成员变量 m\_Edit 和 m\_Grid。

(3) 处理“...”按钮的单击事件,在该事件的处理函数中将数据库文件名添加到列表中,代码如下:

```
void CAttachDlg::OnButton1()
{
 m_Grid.DeleteAllItems(); //清空列表中数据
 CFileDialog file(true,NULL,NULL,OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,
 "All Files (*.*)|*.*||",AfxGetMainWnd()); //构造打开对话框
 if(file.DoModal()==IDOK)
 {
 strText = file.GetPathName(); //获得文件路径
 strName = file.GetFileName(); //获得文件名
 m_Edit.SetWindowText(strText); //显示文件路径
 //设置数据库文件和日志文件名
 str = strName.Left(strName.GetLength()-9);
 CString mdfname="",logname="";
 mdfname.Format("%s_Data.MDF",str);
 logname.Format("%s_Log.LDF",str);
 strmdf = strText;
 strlog = strText;
 strlog.Replace("Data.MDF","Log.LDF");
 //向列表中插入数据
 m_Grid.InsertItem(0,"");
 m_Grid.SetItemText(0,0,mdfname);
 m_Grid.SetItemText(0,1,strmdf);
 m_Grid.InsertItem(1,"");
 m_Grid.SetItemText(1,0,logname);
 m_Grid.SetItemText(1,1,strlog);
 }
}
```

(4) 处理“附加”按钮的单击事件,在该事件中完成数据库的附加操作,代码如下:

```
void CAttachDlg::OnButton2()
{
 OnInitADOConn();
 _bstr_t sql;
 sql = "EXEC sp_attach_db @dbname = N"+str+" ,@filename1 = N"+
 strmdf+" , @filename2 = N"+strlog+""; //设置数据库附加语句
 m_pConnection->Execute(sql,NULL,adCmdText); //执行附加数据库操作
 ExitConnect();
 MessageBox("附加数据库完成","附加数据库",MB_ICONASTERISK);
}
```

### 举一反三

根据本实例,读者可以:

- 在程序中批量附加数据库。

## 实例 293 分离数据库

本实例是一个提高效率的程序

实例位置: 光盘\mingrisoft\08\293

### 实例说明

本实例实现的是通过程序分离 SQL Server 数据库。运行程序,在列表选择一个数据库,双击该列表项,在提示窗口中单击“确定”按钮完成数据库的分离,如图 8.44 所示。

### 技术要点

本实例使用 sp\_detach\_db 存储过程来实现数据库的分离,sp\_detach\_db 存储过程的语法如下:

```
sp_detach_db [@dbname =] 'dbname'
[, [@skipchecks =] 'skipchecks']
```



图 8.44 分离数据库

参数说明:

- `[@dbname =] 'dbname'`: 要分离的数据库名称。dbname 的数据类型为 sysname, 默认值为 NULL。
- `[@skipchecks =] 'skipchecks'`: skipchecks 的数据类型为 nvarchar(10), 默认值为 NULL。如果为 true, 则跳过 UPDATE STATISTICS。如果为 false, 则运行 UPDATE STATISTICS。对于要移动到只读媒体上的数据库, 此选项很有用。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加一个列表框控件, 为列表框控件添加成员变量 `m_Database`。
- (3) 在对话框初始化时, 将服务器中的数据库添加到列表中, 代码如下:

```
BOOL CDetachDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...// 系统代码省略
 OnInitADOConn(); //连接数据库
 _bstr_t bstrSQL;
 bstrSQL = "select name from sysdatabases"; //设置查询语句
 m_pRecordset.CreateInstance(__uuidof(Recordset));
 m_pRecordset->Open(bstrSQL, m_pConnection.GetInterfacePtr(), adOpenDynamic,
 adLockOptimistic, adCmdText); //打开记录集
 while(!m_pRecordset->adoEOF)
 {
 m_Database.AddString((char*)(_bstr_t)m_pRecordset->GetCollect("name")); //向列表中插入数据库名称
 m_pRecordset->MoveNext(); //向下移动记录集指针
 }
 ExitConnect(); //断开数据库连接

 return TRUE;
}
```

- (4) 处理列表框控件的双击事件, 在该事件中进行分离数据库操作, 代码如下:

```
void CDetachDlg::OnDbClickList1()
{
 CString strname, database;
 m_Database.GetText(m_Database.GetCurSel(), database); //获得选择的数据库名称
 strname.Format("确定分离数据库%s吗?", database);
 if(MessageBox(strname, "分离数据库", MB_OKCANCEL|MB_ICONQUESTION) == IDOK) //弹出提示框
 {
 OnInitADOConn(); //连接数据库
 CString sql;
 sql = "EXEC sp_detach_db '"+database+"', 'TRUE'"; //设置SQL语句
 m_pConnection->Execute((_bstr_t)sql, NULL, adCmdText); //执行分离命令
 m_pConnection->Close(); //断开数据库连接
 MessageBox("分离数据库完成!", "分离数据库", MB_ICONASTERISK);
 }
}
```

## 举一反三

根据本实例, 读者可以:

- 批量分离数据库。

增 查 查 查 ( 8 )

## 第 9 章

# SQL 查询相关技术

- 通用查询
- 周期、日期查询
- 比较、逻辑、重复记录查询
- 排序、分组统计
- 聚集函数

Visual C++

增 查 查 查

PDG



## 9.1 通用查询

在数据库中，数据查询是通过 SELECT 语句来完成的。SELECT 语句可以在数据库中按用户的要求和提供的限定条件进行数据检索，并将查询结果以表格的形式返回。下面通过实例来具体介绍用 SELECT 子句完成数据查询的方法及技巧。

### 实例 294 SELECT 语句的应用方法

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\09\294

#### 实例说明

SQL 语言和基于 SQL 的关系数据库系统是计算机领域中最重要基础技术之一。SQL 是结构化查询语言 Structured Query Language 的缩写。在对数据库操作的过程中，使用 SQL 语句会高效地完成指定的作业，在 Visual C++ 中对 SQL 也具有良好支持。当使用 ADO 控件进行数据库操作时，使用 SQL 语句是首选方案。运行程序，在“列名”下拉列表框中选择“性别”，在“内容”编辑框中输入“男”，然后单击“查询”按钮，查询结果将显示在表格中，如图 9.1 所示。

#### 技术要点

SELECT 语句是 SQL 语句的核心，主要用于查询数据库并检索匹配已指定条件的选择数据。

SELECT 语句的语法如下：

```
SELECT [predicate]{*[table.][field [, [table.][field2[,...]]]} [AS alias1 [, alias2[,...]]]
FROM tableexpression [...][In externaldatabase]
[WHERE...]
[GROUP BY...]
[HAVING...]
[ORDER BY...]
[With Owner Access Option]
```

参数说明：

- predicate：包括了 ALL、DISTINCT、DISTINCTROW 和 TOP 等语句，可以利用这样的语句去限制查询后所得的结果。
- table：针对被选择出的记录的字段指定表格的名称。
- field1、field2：想要读取数据的字段名称，如果包含了一个以上的字段，则依照列出的顺序来读取数据。
- alias1、alias2：用来替代表格实际字段名称的别名。
- tableexpression：表格名称或包含用户所想要的数据的表格。
- externaldatabase：若使用的不是目前的数据库，则将其名称定义在 externaldatabase 当中。

注意：FROM 是惟一必需的子句。字段与字段之间以“，”分割，最后一个字段除外。

本实例的实现过程如图 9.2 所示。

#### 实现过程

- (1) 新建一个基于对话框的应用程序。



图 9.1 SQL 语句的应用方法

| 编号  | 姓名  | 性别 | 职业  | 年龄 | 工资      | 是否离职 |
|-----|-----|----|-----|----|---------|------|
| 001 | 李XX | 女  | 服装部 | 25 | 2500.00 | 否    |
| 002 | 陆XX | 男  | 保安部 | 23 | 1800.00 | 否    |
| 003 | 姜XX | 男  | 保安部 | 22 | 2300.00 | 否    |
| 004 | 花X  | 男  | 服装部 | 25 | 1700.00 | 否    |
| 005 | 海X  | 女  | 服装部 | 27 | 1900.00 | 否    |
| 006 | 龙X  | 女  | 服装部 | 21 | 1500.00 | 否    |
| 007 | 雷X  | 女  | 服装部 | 20 | 2100.00 | 否    |
| 008 | 香XX | 男  | 销售部 | 28 | 1650.00 | 否    |
| 009 | 士X  | 男  | 保安部 | 26 | 1550.00 | 否    |
| 010 | 柯X  | 男  | 销售部 | 20 | 1890.00 | 否    |
| 011 | 周X  | 男  | 销售部 | 22 | 2700.00 | 否    |
| 012 | 路X  | 男  | 销售部 | 24 | 2000.00 | 否    |

图 9.2 实现过程

(2) 在菜单中选择 Project/Add To Project/Components and Controls 命令，弹出 Components and Controls Gallery 窗口，双击窗口中的 Registered ActiveX Controls 文件夹，找到 Microsoft ADO Data Control6.0(SP4)(OLEDB)选项，双击它取默认值，添加控件，单击“Close”按钮，ADO Data 控件就添加到控件面板中了，用同样的方法找到 Microsoft DataGrid Control6.0(SP5)(OLEDB)选项，添加 DataGrid 控件。

(3) 在窗体上添加一个组合框控件、一个编辑框控件、一个 ADO Data 控件和一个 DataGrid 控件，为 ADO Data 控件添加成员变量 m\_adodc。

(4) 主要程序代码如下：

```
void CSqlqueryDlg::OnOK()
{
 UpdateData(true);
 CString str;
 m_combo.GetLBText(m_combo.GetCurSel(),str); //获得组合框中数据
 m_adodc.SetRecordSource("select*from kjbdsjk where "+str+"="+m_edit+""); //设置数据源
 m_adodc.Refresh();
}
```

### 举一反三

根据本实例，读者可以：

- 查询指定员工的信息。

## 实例 295 SQL 语句的模糊查询

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\09\295

### 实例说明

模糊查询是数据查询中经常用到的一种查询方式。在查询的时候，经常会忘记要查询的全部内容，这时就需要运用模糊查询。本实例实现了模糊查询功能，在“列名”下拉列表框中选择“姓名”，在“LIKE”编辑框中输入查询的部分内容，单击“查询”按钮，在数据表中将显示符合该条件的所有信息，结果如图 9.3 所示。

### 技术要点

LIKE 关键字搜索与指定模式匹配的字符串、日期或时间值。LIKE 关键字使用常规表达式包含值所要匹配的模式。模式包含要搜索的字符串，字符串中可包含 4 种通配符的任意组合，SQL 通配符如表 9.1 所示。

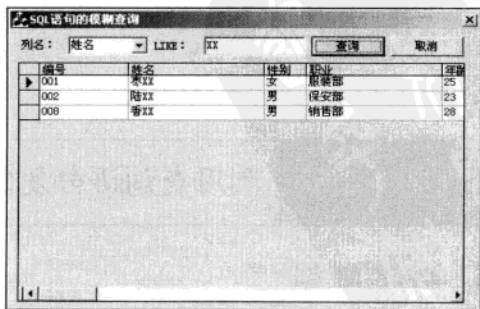


图 9.3 SQL 语句的模糊查询

表 9.1 SQL 通配符表

| 通 配 符 | 描 述               |
|-------|-------------------|
| %     | 代表零个或多个字符         |
| _     | 代表任何单一字符          |
| []    | 在指定区域或集合内的任何单一字符  |
| [^]   | 不在指定区域或集合内的任何单一字符 |

注意：在对字符串进行模糊查询的时候，不可以在字符串中任意加空格，且输入的关键字必须是所查询字符串的子字符串。

本实例的实现过程如图 9.4 所示。

| 编号  | 姓名  | 性别 | 职业  | 年龄 | 工资      | 是否离职 |
|-----|-----|----|-----|----|---------|------|
| 001 | 李XX | 女  | 服装部 | 25 | 2500.00 | 否    |
| 002 | 陆XX | 男  | 保安部 | 23 | 1800.00 | 否    |
| 003 | 美堂X | 男  | 保安部 | 22 | 2300.00 | 否    |
| 004 | 花X  | 男  | 服装部 | 25 | 1700.00 | 否    |
| 005 | 海X  | 女  | 服装部 | 27 | 1900.00 | 否    |
| 006 | 龙X  | 女  | 服装部 | 21 | 1500.00 | 否    |
| 007 | 雪X  | 女  | 服装部 | 20 | 2100.00 | 否    |
| 008 | 香XX | 男  | 销售部 | 28 | 1650.00 | 否    |
| 009 | 士X  | 男  | 保安部 | 26 | 1550.00 | 否    |
| 010 | 柯X  | 男  | 销售部 | 20 | 1890.00 | 否    |
| 011 | 周X  | 男  | 销售部 | 22 | 2700.00 | 否    |
| 012 | 陆X  | 男  | 销售部 | 24 | 2000.00 | 否    |

| 编号  | 姓名  | 性别 | 职业  | 年龄 | 工资      | 是否离职 |
|-----|-----|----|-----|----|---------|------|
| 001 | 李XX | 女  | 服装部 | 25 | 2500.00 | 否    |
| 002 | 陆XX | 男  | 保安部 | 23 | 1800.00 | 否    |
| 008 | 香XX | 男  | 销售部 | 28 | 1650.00 | 否    |

图 9.4 实现过程

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个组合框控件、一个编辑框控件、一个 ADO Data 控件和一个 DataGrid 控件，为 ADO Data 控件添加成员变量 m\_adodc。
- (3) 主要程序代码如下：

```
void CMohuqueryDlg::OnOK()
{
 UpdateData(true);
 CString str;
 m_combo.GetLBText(m_combo.GetCurSel(),str);
 m_adodc.SetRecordSource("select*from kjbdsjk where "+str+" like '%" +m_edit+"%'"); //获得组合框中数据
 m_adodc.Refresh(); //设置数据源
}
```

## 举一反三

根据本实例，读者可以：

- 用 “-” 进行查询；
- 用 “[ ]” 进行查询。

## 实例 296 利用查询语句复制表结构

本实例是一个提高效率的程序

实例位置：光盘\mingrisoft\09\296

## 实例说明

利用查询语句复制表结构就是利用 SELECT INTO 语句生成一个相同表结构的空表。运行程序，销售表中数据将显示在表格中，如图 9.5 所示。单击“复制表结构”按钮，表格中将显示与销售表结构相同的新生成的空表，如图 9.6 所示。

| 书号  | 书名              | 作者   | 出版社  | 原价 | 销售数量 | 仓库   |
|-----|-----------------|------|------|----|------|------|
| 007 | VC数据库系统开发完全手册   | 明日科技 | 人民邮电 | 52 | 31   | 南京仓库 |
| 009 | VC管理信息系统完整项目实例  | 明日科技 | 人民邮电 | 40 | 64   | 南京仓库 |
| 005 | JSP信息开发实例精选     | 明日科技 | 机械工业 | 35 | 25   | 南京仓库 |
| 004 | VC技术大全          | 明日科技 | 人民邮电 | 60 | 36   | 南京仓库 |
| 001 | Delphi数据库开发实例解析 | 明日科技 | 机械工业 | 43 | 46   | 南京仓库 |
| 007 | VC数据库系统开发完全手册   | 明日科技 | 人民邮电 | 52 | 14   | 东方仓库 |
| 009 | VC管理信息系统完整项目实例  | 明日科技 | 人民邮电 | 40 | 16   | 东方仓库 |
| 005 | JSP信息开发实例精选     | 明日科技 | 机械工业 | 35 | 31   | 东方仓库 |
| 004 | VC技术大全          | 明日科技 | 人民邮电 | 60 | 42   | 东方仓库 |
| 001 | Delphi数据库开发实例解析 | 明日科技 | 机械工业 | 43 | 31   | 东方仓库 |
| 007 | VC数据库系统开发完全手册   | 明日科技 | 人民邮电 | 52 | 30   | 西门仓库 |
| 009 | VC管理信息系统完整项目实例  | 明日科技 | 人民邮电 | 40 | 34   | 西门仓库 |
| 007 | VC数据库系统开发完全手册   | 明日科技 | 人民邮电 | 52 | 24   | 北斗仓库 |

图 9.5 销售表

| 书号 | 书名 | 作者 | 出版社 | 原价 | 销售数量 | 仓库 |
|----|----|----|-----|----|------|----|
|----|----|----|-----|----|------|----|

图 9.6 具有销售表结构的新表

## 技术要点

使用 SELECT 语句的 INTO 选项可以创建一个临时表来包含 SELECT 语句的结果集。在数据转换期间以及在必须动态地处理可变的源表结构的应用程序中，经常会用到 SELECT INTO 命令。

SELECT INTO 命令的简明语法如下：

```
SELECT Columns
INTO NewTable
FROM DataSourcees
[WHERE Conditions]
```

参数说明：

- Columns：查询的字段列表。
- NewTable：新创建的表名。
- DataSourcees：查询的表名。
- Conditions：查询的条件。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个按钮控件、一个 ADO Data 控件和一个 DataGrid 控件，为 ADO Data 控件添加成员变量 m\_adodc。
- (3) 主要程序代码如下：

```
void CSelectIntoDlg::OnButtoncopy()
{
 m_datagrid.SetCaption("具有销售表结构的新表: copyframe"); //设置DataGrid控件标题
 OnInitADOConn();
 _bstr_t vSQL;
 vSQL="select * into copyframe from xiaoshoubiao where 书号 = NULL"; //设置查询语句
 ExecuteSQL(vSQL); //执行查询语句
 ExitConnect();
 m_adodc.SetRecordSource("select * from copyframe"); //设置数据源
 m_adodc.Refresh();
}
```

## 举一反三

根据本实例，读者可以：

- 将符合条件的记录保存在临时表中。

## 9.2 周期、日期查询

本节将通过几个实例来介绍关于周期、日期的查询方法。



## 实例 297 查询指定时间段的数据

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\09\297

## 实例说明

本实例实现的是在社团信息表中查询指定时间段加入的社员信息。运行程序, 单击“查询”按钮, 即可将在 2005-2-24~2006-9-24 这段时间加入社团的社员信息显示在表格中, 如图 9.7 所示。

## 技术要点

要实现对指定日期时间段数据的查询, 可以在 SQL 语句中使用 BETWEEN 运算符。

BETWEEN 运算符可以用在 WHERE 子句中选取给定范围的列值所在行。

在使用 BETWEEN AND 进行范围查询时, 查询结果中包含 AND 两边的条件值。在当前的查询中包含了 2005-2-24 和 2006-9-24 两天的销售情况。

也可以利用小于等于 ( $\leq$ ) 和大于等于 ( $\geq$ ) 运算符来代替 BETWEEN。使用该运算符所返回的结果和使用 BETWEEN 运算符相同。

本实例的实现过程如图 9.8 所示。

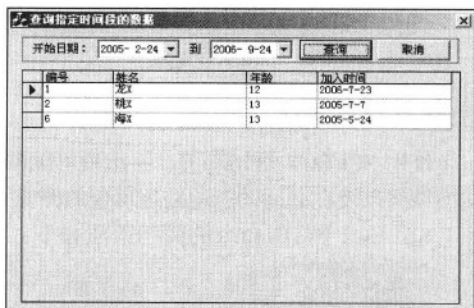


图 9.7 查询指定时间段的数据

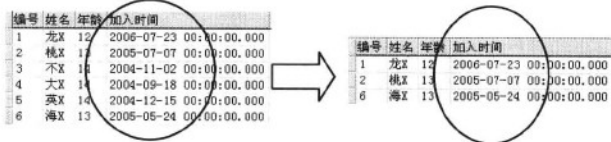


图 9.8 实现过程

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加两个时间控件、一个 ADO Data 控件和一个 DataGrid 控件, 为 ADO Data 控件添加成员变量 m\_adodc。
- (3) 主要程序代码如下:

```
void CZdsjdqueryDlg::OnOK()
{
 UpdateData(true);
 CTime time;
 m_date1.GetTime(time); //获得时间控件中数据
 CString stry, strm, strd, date1, date2;
 stry.Format("%d", time.GetYear());
 strm.Format("%d", time.GetMonth());
 strd.Format("%d", time.GetDay());
 date1 = stry + "-" + strm + "-" + strd;
 m_date2.GetTime(time); //获得时间控件中数据
 stry.Format("%d", time.GetYear());
 strm.Format("%d", time.GetMonth());
 strd.Format("%d", time.GetDay());
 date2 = stry + "-" + strm + "-" + strd;
 //设置数据源
 m_adodc.SetRecordSource("select*from shuzcx where 加入时间>=" + date1 + " and 加入时间<=" + date2 + "");
 m_adodc.Refresh();
}
```

## 举一反三

根据本实例，读者可以：

- 在信息管理系统中查询存在于某一范围内的数据信息；
- 在信息管理系统中利用 NOT BETWEEN 运算符查询不在某一范围内的数据信息情况。

## 实例 298 按月查询数据

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\09\298

## 实例说明

在进行日期数据查询时，经常会因为忘记具体的时间而无法查询到所需的数据。这种情况下，可以通过月来对数据进行查询。运行程序，在下拉列表中选择月份，单击“查询”按钮，即可在数据表中显示查询的结果，如图 9.9 所示。

## 技术要点

要实现按月查询数据，可以使用日期函数 MONTH，该函数的代码如下：

MONTH (date)

MONTH 函数能够将日期时间表达式 date 中的月份返回，返回的月份是以数值 1~12 来表示的，每个数字代表相应的月份。

**注意：**日期时间表达式 date 必须是 datetime 或 smalldatetime 数据类型。

本实例的实现过程如图 9.10 所示。

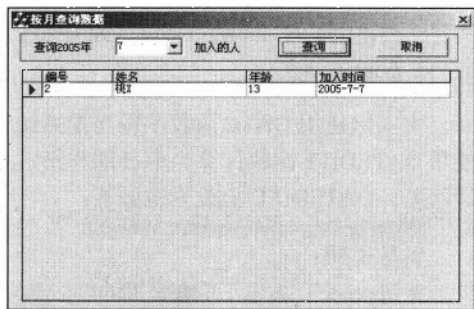


图 9.9 按月查询数据

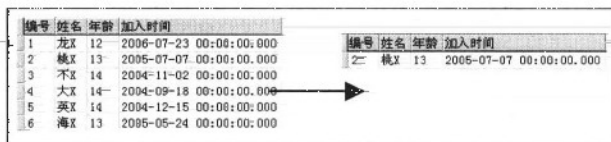


图 9.10 实现过程

## 实现过程

- 新建一个基于对话框的应用程序。
- 在窗体上添加一个时间控件、一个 ADO Data 控件和一个 DataGrid 控件，为 ADO Data 控件添加成员变量 m\_adoc.

(3) 主要程序代码如下：

```
void CMonthqueryDlg::OnOK()
{
 UpdateData(true);
 CString str;
 m_combo.GetLBText(m_combo.GetCurSel(), str); //获得时间控件中数据
 //设置数据源
 m_adoc.SetRecordSource("select*from shuycx where year(加入时间)="+str+" and month(加入时间)="+str+" ");
 m_adoc.Refresh();
}
```

## 举一反三

根据本实例，读者可以：

- 按年份查询。

## 实例 299 在查询中使用日期函数

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\09\299

## 实例说明

在 SQL 语句查询中经常会用到日期函数, 本实例将介绍日期函数的使用方法。运行程序, 单击“查询”按钮, 在数据表中的“出生日期”字段后将自动创建员工的“年龄”字段, 如图 9.11 所示。

## 技术要点

本实例使用 DATE 函数获得当前系统日期, 使用 DATEDIFF 函数计算当前日期与指定日期的时间差, DATEDIFF 函数原型如下:

DATEDIFF ( datepart , startdate , enddate );

参数说明:

- datepart: 规定了应在日期的哪一部分计算差额的参数。datepart 参数的可选值如表 9.2 所示。

表 9.2

datepart 可选值表

| 可 选 值     | 缩 写      | 可 选 值       | 缩 写    |
|-----------|----------|-------------|--------|
| year      | yy, yyyy | Week        | wk, ww |
| quarter   | qq, q    | Hour        | hh     |
| Month     | mm, m    | minute      | mi, n  |
| dayofyear | dy, y    | second      | ss, s  |
| Day       | dd, d    | millisecond | ms     |

- startdate: 计算的开始日期。
- enddate: 计算的结束日期

本实例的实现过程如图 9.12 所示。

| 编号  | 姓名  | 出生日期       | 年龄 |
|-----|-----|------------|----|
| 001 | 徐子陵 | 1980-11-11 | 25 |
| 002 | 寇仲  | 1980-07-14 | 26 |
| 003 | 郭靖  | 1978-03-05 | 28 |
| 004 | 沈落雁 | 1982-06-08 | 24 |
| 005 | 赵敏  | 1985-08-09 | 21 |
| 006 | 袁紫衣 | 1984-12-28 | 22 |
| 007 | 夏雪仪 | 1976-02-14 | 30 |

图 9.12 实现过程

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在窗体上添加一个 ADO Data 控件和一个 DataGrid 控件, 为 ADO Data 控件添加成员变量 m\_adoc。

(3) 主要程序代码如下:

```
void CDatequeryDlg::OnOK()
{
 m_adoc.SetRecordSource("select sh.编号,sh.姓名,sh.出生日期,DateDiff('yyyy',sh.出生日期,DATE()) AS 年龄 from shujubiao as sh"); //设置数据源
 m_adoc.Refresh();
}
```



图 9.11 在查询中使用日期函数

## 举一反三

根据本实例，读者可以：

- 利用日期函数计算员工的工龄；
- 利用日期函数实现网吧计时管理系统。

## 9.3 比较、逻辑、重复记录查询

大小比较、逻辑查询等条件是 SQL 查询技术中较常用的部分。通过下面几个例子的学习，读者可以了解大小比较、逻辑查询等操作的具体应用。

## 实例 300

## NOT 与谓词进行组合条件的查询

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\09\300

## 实例说明

本实例实现的是在员工信息表中查询员工性别为男且员工年龄不在 22~25 岁之间的员工信息。运行程序，选中相应的单选按钮，单击“查询”按钮，即可将员工性别为男且员工年龄不在 22~25 岁之间的员工信息显示在表格中，如图 9.13 所示。

## 技术要点

本实例使用 NOT 与谓词组合所形成的条件来进行查询。

NOT 与谓词进行组合所形成表达式分别是 [NOT] BETWEEN，IS [NOT] NULL，[NOT] IN。

(1) [NOT] BETWEEN。该条件指定值的包含范围，使用 AND 将开始值和结束值分开，语法如下：

```
test_expression [NOT] BETWEEN begin_expression AND end_expression;
```

参数说明：

- NOT BETWEEN：查找用 BETWEEN 指定范围之外的所有行。

用户还需注意，若要指定排除范围，还可以使用大于 (>) 和小于 (<) 运算符来代替 BETWEEN。

(2) IS [NOT] NULL。该式根据所使用的关键字指定对空值或非空值的搜索，如果有任何操作数是 NULL，表达式取值为 NULL。

(3) [NOT] IN。该式根据使用的关键字是包含在列表内还是排除在列表外指定对表达式的搜索，搜索表达式可以是常量或列名，而列表可以是一组常量，但更多情况下是子查询，将值列表放在圆括号内。语法如下：

```
test_expression[NOT] IN
(
 subquery
 expression[, ...n]
)
```

参数说明：

- test\_expression：SQL 表达式。
- subquery：包含某列结果集的子查询，该列必须与 test\_expression 具有相同的数据类型。
- expression[, ...n]：一个表达式列表，用来测试是否匹配，所有的表达式必须和 test\_expression

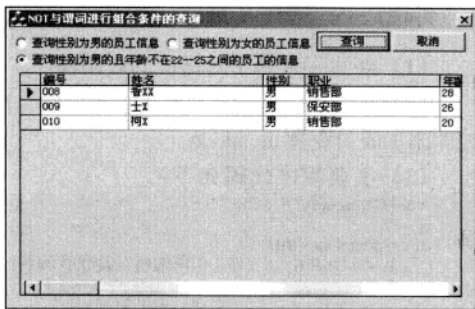


图 9.13 NOT 与谓词进行组合条件的查询



具有相同的数据类型。

该式的返回值是根据 test\_expression 与 subquery 返回的值进行比较, 如果两个值相等, 或与逗号分隔的列表中的任何 expression 相等, 那么结果值就为 TRUE, 否则结果值为 FALSE。

使用 NOT IN 对返回值取反。

本实例的实现过程如图 9.14 所示。



| 编号  | 姓名  | 性别 | 职业  | 年龄 | 工资   | 是否离职 |
|-----|-----|----|-----|----|------|------|
| 001 | 李XX | 女  | 服装部 | 25 | 2500 | 否    |
| 002 | 陆XX | 男  | 保安部 | 23 | 1800 | 否    |
| 003 | 姜XX | 男  | 保安部 | 22 | 2300 | 否    |
| 004 | 花XX | 男  | 服装部 | 25 | 1700 | 否    |
| 005 | 海XX | 女  | 服装部 | 27 | 1900 | 否    |
| 006 | 龙XX | 女  | 服装部 | 21 | 1500 | 否    |
| 007 | 雷XX | 女  | 服装部 | 20 | 2100 | 否    |
| 008 | 曹XX | 男  | 销售部 | 26 | 1650 | 否    |
| 009 | 土XX | 男  | 保安部 | 26 | 1550 | 否    |
| 010 | 柯XX | 男  | 销售部 | 20 | 1690 | 否    |
| 011 | 周XX | 男  | 销售部 | 22 | 2700 | 否    |
| 012 | 路XX | 男  | 销售部 | 24 | 2000 | 否    |

| 编号  | 姓名  | 性别 | 职业  | 年龄 | 工资   | 是否离职 |
|-----|-----|----|-----|----|------|------|
| 008 | 曹XX | 男  | 销售部 | 26 | 1650 | 否    |
| 009 | 土XX | 男  | 保安部 | 26 | 1550 | 否    |
| 010 | 柯XX | 男  | 销售部 | 20 | 1690 | 否    |

图 9.14 实现过程

## 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“NOT 与谓词进行组合条件的查询”。
- (2) 在窗体上添加 3 个单选按钮控件、1 个 ADO Data 控件和 1 个 DataGrid 控件, 为 ADO Data 控件添加成员变量 m\_adodc。
- (3) 主要程序代码如下:

```
void CNotqueryDlg::OnOK()
{
 UpdateData(true);
 switch(radio) //根据单选按钮设置查询条件
 {
 case 0: //查询男员工信息
 m_adodc.SetRecordSource("select*from kjbdsjk where 性别='男'");
 break;
 case 1: //查询女员工信息
 m_adodc.SetRecordSource("select*from kjbdsjk where 性别='女'");
 break;
 case 2: //查询年龄不在22~25的男员工信息
 m_adodc.SetRecordSource("select*from kjbdsjk where 性别='男' and 年龄 not between 22 and 25");
 break;
 }
 m_adodc.Refresh();
}
```

## 举一反三

根据本实例, 读者可以:

- 查询编号不在 10~20 之间的所有记录。

## 实例 301 查询时不显示重复记录

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\09\301

## 实例说明

本实例实现的是在图书库存信息表中查询图书情况, 不显示重复的记录。运行程序, 单击“查询”按钮, 即可在表格中显示仓库中剩余图书的情况, 并去除重复记录, 如图 9.15 所示。

## 技术要点

在实现查询时用到了 DISTINCT 关键字, 该关键字用于去除重复记录。

在实现查询操作时, 如果查询的选择列表中包含一个表的主键, 那么每个查询结果中的记录将是惟一的 (因为主键在每一条记录中有一个不同的值)。如果主键不包含在查询结果中, 就可能出现重复记录。使用了 DISTINCT 关键字以后就可以消除重复记录。

本实例的实现过程如图 9.16 所示。

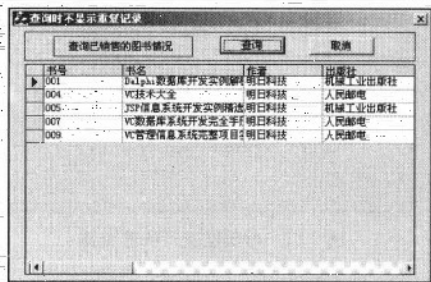


图 9.15 查询时不显示重复记录

| 书号  | 书名               | 作者   | 出版社     |
|-----|------------------|------|---------|
| 001 | Delphi数据库开发实例解析  | 明日科技 | 机械工业出版社 |
| 001 | Delphi数据库开发实例解析  | 明日科技 | 机械工业出版社 |
| 004 | VC技术大全           | 明日科技 | 人民邮电    |
| 004 | VC技术大全           | 明日科技 | 人民邮电    |
| 005 | JSP信息系统开发实例精选    | 明日科技 | 机械工业出版社 |
| 005 | JSP信息系统开发实例精选    | 明日科技 | 机械工业出版社 |
| 007 | VC数据库系统开发完全手册    | 明日科技 | 人民邮电    |
| 007 | VC数据库系统开发完全手册    | 明日科技 | 人民邮电    |
| 007 | VC数据库系统开发完全手册    | 明日科技 | 人民邮电    |
| 007 | VC数据库系统开发完全手册    | 明日科技 | 人民邮电    |
| 009 | VC管理信息系统完整项目实例剖析 | 明日科技 | 人民邮电    |
| 009 | VC管理信息系统完整项目实例剖析 | 明日科技 | 人民邮电    |
| 009 | VC管理信息系统完整项目实例剖析 | 明日科技 | 人民邮电    |

图 9.16 本实例的实现过程

### 注意:

- (1) 如果省略了 DISTINCT 关键字, 查询结果中不会消除重复的记录。也可以指定 ALL 关键字来明确指示要保留重复记录, 但这是不必要的, 因为这是默认的行为。
- (2) DISTINCT 关键字并不是指某一行, 而是指不重复 SELECT 输出的所有列。这一点十分重要, 其作用是防止相同的行出现在一个查询结果的输出中。
- (3) DISTINCT 是 SUM、AVG 和 COUNT 函数的可选关键字。如果使用 DISTINCT 关键字, 那么在计算总和、平均值或计数之前, 先消除重复的值。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个 ADO Data 控件和一个 DataGrid 控件, 为 ADO Data 控件添加成员变量 m\_adodc。
- (3) 主要程序代码如下:

```
void CBXscfjDlg::OnOK()
{
 UpdateData(true);
 //设置数据源
 m_adodc.SetRecordSource("select distinct 书号,书名,作者,出版社 from chongfujilu order by 书号");
 m_adodc.Refresh();
}
```

## 举一反三

根据本实例, 读者可以:

- 在销售表中查询所有商品的销售情况。

## 9.4 排序、分组统计

在查询统计中经常会遇到对查询结果进行排序、分组的情况, 在这种情况下需用到 GROUP BY 子句和 ORDER BY 子句。下面介绍几个实现排序、分组统计的例子。

## 实例 302 对数据进行降序查询

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\09\302

## 实例说明

本实例实现的是在社团信息表中对年龄进行降序排序。运行程序, 单击“查询”按钮, 即可将按降序排序后的年龄信息显示在表格中, 如图 9.17 所示。

## 技术要点

在实现对数据进行降序排序查询时用到了 ORDER BY 子句和 DESC 关键字。其实现方法是在 SQL 语句的查询语句中添加“ORDER BY 年龄 DESC”子句即可。

ORDER BY 子句的作用是分类输出, 并且根据表中包含的一列或多列的表达式将输出按升序或降序排列, 不改变数据库中的行的顺序。ORDER BY 简单改变查询输出的显示顺序。

ORDER BY 子句的语法如下:

```
SELECT ...
ORDER BY expression [ASC|DESC],...
WHERE
```

参数说明:

- expression: 指定输出时排序的字段。
- [ASC|DESC]: 可选项, 代表升序或者降序。
- ...: 指可以有多个的分类表达式, 并且每一个表达式都可以分别为升序或者降序。

排序表达式中可包括未出现在 SELECT 子句选择列表中的列名。如果在 SELECT 子句中使用了 DISTINCT 关键字, 或查询语句中包含 UNION 运算符, 则排序列必须包含在 SELECT 子句选择列表中。

本实例的实现过程如图 9.18 所示。

| 编号 | 姓名 | 年龄 | 加入时间                    |
|----|----|----|-------------------------|
| 1  | 龙X | 12 | 2006-07-23 00:00:00.000 |
| 2  | 桃X | 13 | 2005-07-07 00:00:00.000 |
| 3  | 不X | 14 | 2004-11-02 00:00:00.000 |
| 4  | 大X | 14 | 2004-09-18 00:00:00.000 |
| 5  | 英X | 14 | 2004-12-15 00:00:00.000 |
| 6  | 海X | 13 | 2005-05-24 00:00:00.000 |

图 9.18 实现过程

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个 ADO Data 控件和一个 DataGrid 控件, 为 ADO Data 控件添加成员变量 m\_adocdc。

(3) 主要程序代码如下:

```
void CJXQueryDlg::OnOK()
{
 UpdateData(true);
 m_adocdc.SetRecordSource("select*from shuzcx order by 年龄 desc"); //设置数据源
 m_adocdc.Refresh();
}
```

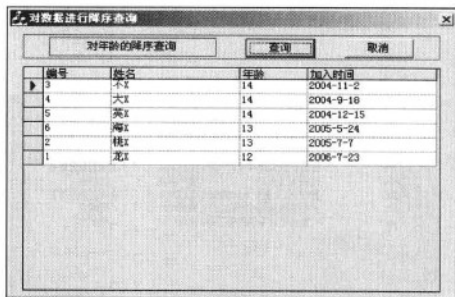


图 9.17 对数据进行降序查询

## 举一反三

根据本实例，读者可以：

- 对销售员的业绩进行降序排列；
- 对学生的考勤天数进行升序排列。

## 实例 303 对数据进行多条件排序

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\09\303

## 实例说明

本实例实现的是对数据表中的数据进行多条件排序。运行程序，单击“查询”按钮，即可将数据表中的数据信息按年龄升序、加入时间降序排列后显示在表格中，如图 9.19 所示。

## 技术要点



图 9.19 对数据进行多条件排序

利用 SQL 语句的 ORDER BY 语句可以实现按多种条件排序数据的功能，语法如下：

[ORDER BY { order\_by\_expression [ASC|DESC]} [...n]];

参数说明：

- order\_by\_expression：指定要排序的列。可以将排序列指定为列名或列的别名（可由表名或视图名限定）或表达式，或者指定为代表选择列表内的名称、别名或表达式的位置的负整数，可指定多个排序列。ORDER BY 子句可包括未出现在此选择列表中的项目。然而，如果指定 SELECT DISTINCT，或者如果 SELECT 语句包含 UNION 运算符，则排序列必定出现在选择列表中。
- ASC：指定按递增顺序进行排序。
- DESC：指定按递减顺序进行排序。

对 ORDER BY 子句中的项目数没有限制。然而，对于排序操作所需的中间级工作表的大小有 8 060 字节的限制。这限制了在 ORDER BY 子句中指定的列的合计大小。

空值被视为最低的可能值。

本实例的实现过程如图 9.20 所示。

**注意：**在 ORDER BY 子句中不能使用 ntext、text 和 image 数据类型的列。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个 ADO Data 控件和一个 DataGrid 控件，为 ADO Data 控件添加成员变量 m\_adodc。
- (3) 主要程序代码如下：

| 编号 | 姓名 | 年龄 | 加入时间                    |
|----|----|----|-------------------------|
| 1  | 龙X | 12 | 2006-07-23 00:00:00.000 |
| 2  | 桃X | 13 | 2005-07-07 00:00:00.000 |
| 3  | 不X | 14 | 2004-11-02 00:00:00.000 |
| 4  | 大X | 14 | 2004-09-18 00:00:00.000 |
| 5  | 英X | 14 | 2004-12-15 00:00:00.000 |
| 6  | 海X | 13 | 2005-05-24 00:00:00.000 |

| 编号 | 姓名 | 年龄 | 加入时间                    |
|----|----|----|-------------------------|
| 1  | 龙X | 12 | 2006-07-23 00:00:00.000 |
| 2  | 桃X | 13 | 2005-07-07 00:00:00.000 |
| 3  | 不X | 14 | 2005-05-24 00:00:00.000 |
| 5  | 英X | 14 | 2004-12-15 00:00:00.000 |
| 3  | 不X | 14 | 2004-11-02 00:00:00.000 |
| 4  | 大X | 14 | 2004-09-18 00:00:00.000 |

图 9.20 实现过程



```
void CDuotjpxDlg::OnOK()
{
 UpdateData(true);
 m_adodc.SetRecordSource("select * from shuzcx order by 年龄,加入时间 desc"); //设置数据源
 m_adodc.Refresh();
}
```

## 举一反三

- 根据本实例，读者可以：
- 对数据进行分组排序。

## 9.5 聚集函数

聚集函数专门应用于 SELECT 命令语句中，以便使用户迅速获得数据表的数据日志的相关统计信息。聚集函数可以在 SELECT 命令的字段行、COMPUTE 参数、COMPUTE BY 参数、GROUP BY 参数、HAVING 参数中使用。

### 实例 304

利用聚集函数 SUM 对销售额进行汇总

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\09\304

### 实例说明

本实例实现的功能是在图书销售表中查询按书名、书号统计销售金额。运行程序，单击“查询”按钮，即可将图书的销售情况显示在表格中，如图 9.21 所示。

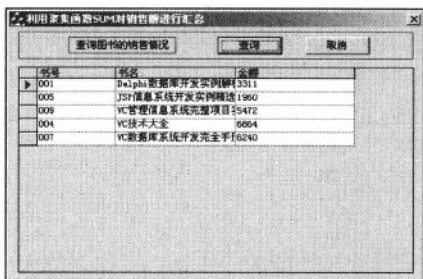


图 9.21 利用聚集函数 SUM 对销售额进行汇总

### 技术要点

本实例利用聚集函数 SUM 对销售额进行汇总。SUM 聚集函数主要用于返回表达式中所有值的和，或只返回 DISTINCT 值。SUM 聚集函数只能用于数据类型是数字的列，NULL 值将被忽略。

SUM 函数的语法形式如下：

SUM ( [ALL|DISTINCT] expression ) ;

参数说明：

- ALL：对所有的值进行聚集函数运算，ALL 是默认设置。
- DISTINCT：指定 SUM 返回惟一值的和。
- expression：是常量、列或函数，或者是算术、按位与字符串等运算符的任意组合。expression 是精确数字或近似数字数据类型分类（BIT 数据类型除外）的表达式。不允许使用聚集函数和子查询。

本实例的实现过程如图 9.22 所示。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个 ADO Data 控件和一个 DataGrid 控件，为 ADO Data 控件添加成员变量 m\_adodc。

| 书号   | 书名                   | 出版社     | 金额    | 出版社名称   |
|------|----------------------|---------|-------|---------|
| 0007 | VC数据库系统开发完全手册        | 机械工业出版社 | 52.00 | 机械工业出版社 |
| 0009 | VC管理信息系统完整项目15472    | 机械工业出版社 | 48.00 | 机械工业出版社 |
| 0005 | JSP信息开发实例精解1960      | 机械工业出版社 | 36.00 | 机械工业出版社 |
| 0004 | VC技术大全 5684          | 机械工业出版社 | 52.00 | 机械工业出版社 |
| 0001 | Delphi数据库开发实例精解13311 | 机械工业出版社 | 48.00 | 机械工业出版社 |
| 0006 | VC数据库系统开发完全手册        | 机械工业出版社 | 42.00 | 机械工业出版社 |
| 0008 | VC管理信息系统完整项目15472    | 机械工业出版社 | 48.00 | 机械工业出版社 |
| 0003 | JSP信息开发实例精解1960      | 机械工业出版社 | 36.00 | 机械工业出版社 |
| 0004 | VC技术大全 5684          | 机械工业出版社 | 52.00 | 机械工业出版社 |
| 0001 | Delphi数据库开发实例精解13311 | 机械工业出版社 | 48.00 | 机械工业出版社 |
| 0009 | VC管理信息系统完整项目15472    | 机械工业出版社 | 48.00 | 机械工业出版社 |
| 0007 | VC数据库系统开发完全手册        | 机械工业出版社 | 52.00 | 机械工业出版社 |

图 9.22 实现过程

(3) 主要程序代码如下:

```
void CSumqueryDlg::OnOK()
{
 //设置数据源
 m_adoc.SetRecordSource("select 书号,书名,sum(单价*销售数量)as 金额 from xiaoshoubiao group by 书号,书名 ");
 m_adoc.Refresh();
}
```

### 举一反三

根据本实例,读者可以:

- 在聚集和行聚集中使用 SUM 函数。

### 实例 305

### 利用聚集函数 AVG 求某班学生的平均年龄

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\09\305

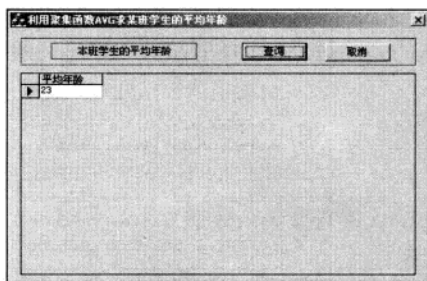


图 9.23 利用聚集函数 AVG 求某班学生的平均年龄

### 实例说明

本实例利用聚集函数 AVG 查询统计学生的平均年龄。运行程序,单击“统计”按钮,即可将学生的平均年龄显示在表格中,如图 9.23 所示。

### 技术要点

AVG 聚集函数主要用于返回组中值的平均值,空值将被忽略。

AVG 函数的语法形式如下:

AVG ([ALL|DISTINCT] expression ) ;

参数说明:

- ALL: 对所有的值进行聚集函数运算,是默认设置。
- DISTINCT: 指定 AVG 操作只使用每个值的惟一实例,而不管该值出现了多少次。
- expression: 精确数字或近似数字数据类型类别的表达式 (BIT 数据类型除外),不允许使用聚集函数和子查询。

本实例的实现过程如图 9.24 所示。

| 编号  | 姓名 | 性别 | 年龄 |
|-----|----|----|----|
| 001 | 李红 | 女  | 25 |
| 002 | 陈红 | 男  | 23 |
| 003 | 吴红 | 男  | 22 |
| 004 | 王红 | 男  | 25 |
| 005 | 李红 | 女  | 27 |
| 006 | 王红 | 女  | 21 |
| 007 | 李红 | 女  | 28 |
| 008 | 李红 | 男  | 28 |
| 009 | 王红 | 男  | 26 |
| 010 | 李红 | 男  | 20 |
| 011 | 李红 | 男  | 22 |
| 012 | 陈红 | 男  | 24 |

图 9.24 实现过程

### 实现过程

- 新建一个基于对话框的应用程序。
- 在窗体上添加一个 ADO Data 控件和一个 DataGrid 控件,为 ADO Data 控件添加成员变量 m\_adoc。

(3) 主要程序代码如下:

```
void CAVGqueryDlg::OnOK()
{
 m_adoc.SetRecordSource("select avg(年龄) as 平均年龄 from kjbdsjk"); //设置数据源
 m_adoc.Refresh();
}
```

### 举一反三

根据本实例,读者可以:

- 在学生成绩表中求学生的平均成绩。

## 实例 306

利用聚集函数 COUNT 求日  
销售额大于某值的商品数

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\09\306

## 实例说明

COUNT 函数用于返回组中项目的数量。本实例利用 COUNT 函数实现在销售信息表中查询销售额大于 1500 的商品数量。运行程序, 单击“查询”按钮, 即可在表格中显示出销售额大于 1500 的商品数, 如图 9.25 所示。

## 技术要点

图 9.25 利用聚集函数 COUNT 求日销售额大于某值的商品数

本实例使用 COUNT 聚集函数求日销售额大于某值的商品数。COUNT 函数用于返回组中项目的数量。

COUNT 函数的语法形式如下:

COUNT([ALL|DISTINCT]expression[\*])

参数说明:

- ALL: 是默认设置, 如果没有参数, 系统对所有的值进行聚集函数运算。
- DISTINCT: 指定 COUNT 返回惟一非空值的数量。
- expression: 一个表达式, 在 SQL Server 中的类型是除 uniqueidentifier、text、image 或 ntext 之外的任何类型, 不允许使用聚集函数和子查询。
- \*: 指定应该计算所有行以返回表中行的总数。COUNT (\*) 不需要任何参数, 而且不能与 DISTINCT 一起使用。COUNT (\*) 不需要 expression 参数, 因为根据定义, 该函数不使用有关任何特定列的信息。COUNT (\*) 返回指定表中行的数量而不消除副本。它对每行分别进行计数, 包括含有空值的行。
- COUNT(\*): 返回组中项目的数量, 这些项目包括 NULL 值和副本。
- COUNT (ALL expression): 对组中的每一行都计算 expression 并返回非空值的数量。
- COUNT (DISTINCT expression): 对组中的每一行都计算 expression 并返回惟一非空值的数量。

本实例的实现过程如图 9.26 所示。



| 商品名称    | 销售日期       | 销售额     | 商品数 |
|---------|------------|---------|-----|
| 001 鼠标  | 2009-02-28 | 2500.00 | 1   |
| 002 键盘  | 2009-02-28 | 800.00  | 1   |
| 003 显示器 | 2009-03-01 | 1200.00 | 1   |
| 004 打印机 | 2009-03-01 | 900.00  | 1   |
| 005 CPU | 2009-03-01 | 1100.00 | 1   |
| 006 主板  | 2009-03-01 | 700.00  | 1   |
| 007 内存  | 2009-03-01 | 1300.00 | 1   |

图 9.26 实现过程

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在窗体上添加一个 ADO Data 控件和一个 DataGrid 控件, 为 ADO Data 控件添加成员变量 m\_adodc。

(3) 主要程序代码如下:

```
void CCOUNTQueryDlg::OnOK()
{
 //设置数据源
 m_adodc.SetRecordSource("select count(all 日期) as 商品数 from daojubiao where 销售额 >1500");
 m_adodc.Refresh();
}
```

## 举一反三

根据本实例, 读者可以:

- 求月销售数量最多的商品。

第 10 章

# 第 10 章

## 打印与报表技术



- 基础打印
- 打印图片
- 打印单据
- 控制打印
- 打印预览

Visual C++





## 10.1 基础打印

人们在日常工作中经常会用到报表打印,本节将通过几个实例分别介绍基于文档/视图和对话框的报表打印。

## 实例 307

## 基于文档/视图结构的打印

这是一个可以提高基础性能的实例

实例位置:光盘\mingrisoft\10\307

## 实例说明

Visual C++中文档/视图应用程序封装了打印功能。因此,在文档/视图应用程序中可以方便的实现打印功能。本例演示了利用文档/视图应用程序进行打印,效果如图 10.1 所示。



图 10.1 基于文档/视图结构的打印

## 技术要点

视图类 CView 提供了一个 OnPrint 虚拟方法,该方法在打印或预览每一页时由框架调用。其语法如下:

```
virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
```

参数说明:

- pDC 是打印机设备上下文指针。
- pInfo 是一个 CPrintInfo 类指针,记录了当前打印信息。

程序中可以在该方法中利用 pDC 参数输出打印的数据,但是有一点需要注意,就是打印机的分辨率通常比屏幕分辨率高出许多。假如同样的一条输出语句,在屏幕上输出为 1 厘米,在打印机上可能是很短的一段距离。因此,在输出打印信息时,需要设置一定的输出比率。可以利用 CDC 的 GetDeviceCaps 方法间接获取打印机与屏幕的分辨率比率。该方法的语法如下:

```
int GetDeviceCaps(int nIndex) const;
```

参数说明:

- nIndex 标识方法返回信息的类型,本例将其设置为 LOGPIXELSX 或 LOGPIXELSY。
- LOGPIXELSX 是当前 CDC 对象,表示设备水平方向每英寸像素数。
- LOGPIXELSY 是当前 CDC 对象,表示设备垂直方向每英寸像素数。

当窗口的 CDC 对象调用 GetDeviceCaps(LOGPIXELSX)方法时返回的是屏幕水平方向每英寸的

像素数量。当打印机的 CDC 对象调用 GetDeviceCaps(LOGPIXELSX)方法时返回的是打印机水平方向每英寸的像素数量。将两种方法返回的数据相除就可以获得打印机与屏幕的分辨率比率了。

通常可以在 CView 类的 OnDraw 方法中获取屏幕每英寸的像素数量,代码如下:

```
screenx = pDC->GetDeviceCaps(LOGPIXELSX);
screeny = pDC->GetDeviceCaps(LOGPIXELSY);
```

在 CView 类的 OnBeginPrinting 方法中获得打印机每英寸的像素数量,代码如下:

```
printx = pDC->GetDeviceCaps(LOGPIXELSX);
printy = pDC->GetDeviceCaps(LOGPIXELSY);
xrate = (double)printx / screenx; //确定打印机与屏幕的比率
yrate = (double)printy / screeny;
```

## 实现过程

- (1) 新建一个文档/视图结构的应用程序。
- (2) 创建一个对话框类,本例为 CMainDlg,用于提供打印的数据。
- (3) 在对话框中添加按钮和列表视图控件。
- (4) 在对话框初始化时向列表视图控件中添加打印的数据,代码如下:

```
BOOL CMainDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 //add columns for list
 m_list.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_GRIDLINES
|LVS_EX_FULLROWSELECT);
```

```
 m_list.InsertColumn(100,"姓名",LVCFMT_LEFT,100);
 m_list.InsertColumn(100,"语文",LVCFMT_LEFT,70);
 m_list.InsertColumn(100,"数学",LVCFMT_LEFT,70);
 m_list.InsertColumn(100,"英语",LVCFMT_LEFT,70);
 m_list.InsertColumn(100,"政治",LVCFMT_LEFT,70);
 m_list.InsertColumn(100,"历史",LVCFMT_LEFT,70);
```

```
 CString temp;
 int grade;
 //add data to list
 for (int i = 0;i<10;i++)
 {
 m_list.InsertItem(i,"");
 for (int c = 1;c<6;c++)
 {
 grade = c*2+80+i;
 temp.Format("%d",grade);
 m_list.SetItemText(i,c,temp);
 }
 }
```

```
 m_list.SetItemText(0,0,"王平");
 m_list.SetItemText(1,0,"李可");
 m_list.SetItemText(2,0,"张红");
 m_list.SetItemText(3,0,"周亮");
 m_list.SetItemText(4,0,"孙军");
 m_list.SetItemText(5,0,"刘海");
 m_list.SetItemText(6,0,"王中华");
 m_list.SetItemText(7,0,"宋可平");
 m_list.SetItemText(8,0,"张男");
 m_list.SetItemText(9,0,"李菊");
```

```
 return TRUE;
```

- (5) 在应用程序初始化时显示 CMainDlg 对话框,代码如下:

```
BOOL CDocViewPrintApp::InitInstance()
{
 AfxEnableControlContainer();
```

```
#ifdef _AFXDLL
 Enable3dControls();
#else
 Enable3dControlsStatic();
#endif
```

```
SetRegistryKey(_T("Local AppWizard-Generated Applications"));
```

```
LoadStdProfileSettings();
```

```
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
 IDR_MAINFRAME,
 RUNTIME_CLASS(CDocViewPrintDoc),
 RUNTIME_CLASS(CMainFrame),
 RUNTIME_CLASS(CDocViewPrintView));
AddDocTemplate(pDocTemplate);
```

```
CMainDlg m_maindlg;
this->m_pMainWnd = &m_maindlg;
tempdlg = &m_maindlg;
m_maindlg.DoModal();
```

```
return TRUE;
```

(6) 在视图类的 OnPrint 方法中输出打印的数据, 代码如下:

```
void CDocViewPrintView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
```

```
{
 isPreview = TRUE;
 m_titlefont.CreatePointFont((int)xrate*200, "宋体", pDC);
 m_bodyfont.CreatePointFont((int)xrate*100, "宋体", pDC);
 pDC->SelectObject(&m_titlefont);
 CRect m_rect(-leftmargin, 0, pagewidth+rightmargin, pageheight);
 m_rect.DeflateRect(0, (int)20*yrate, 0, 0);
 //绘制标题
 pDC->DrawText("学生成绩单", m_rect, DT_CENTER|DT_SINGLELINE);
 //绘制报表数据
 CMainDlg* pmaindlg = ((CDocViewPrintApp*)(AfxGetApp()))->tempdlg;
 m_rect.DeflateRect(0, (int)20*yrate, 0, 0);
 pDC->SelectObject(&m_bodyfont);
 char* pchar = new char[100];
 LV_COLUMN column;
 column.mask = LVCF_TEXT;
 column.pszText = pchar;
 column.cchTextMax = 100;
 CString str;
 CRect m_temprect((int)(xrate*60), m_rect.top+(int)(xrate*60), (int)(xrate*60)
 +(m_rect.Width()-(int)(xrate*60)*2)/6, m_rect.bottom);
 CRect m_itemrect;
 int width = m_temprect.Width();
 for (int i = 0; i < 6; i++)
 {
 if (pmaindlg->m_list.GetColumn(i, &column))
 str = column.pszText;
 pDC->DrawText(str, m_temprect, DT_LEFT);
 m_itemrect.CopyRect(m_temprect);
 for (int row = 0; row < 10; row++)
 {
 m_itemrect.DeflateRect(0, (int)(xrate*30));
 str = pmaindlg->m_list.GetItemText(row, i);
 pDC->DrawText(str, m_itemrect, DT_LEFT);
 }
 m_temprect.DeflateRect(width+1, 0, 0, 0);
 m_temprect.InflateRect(0, 0, width+1, 0);
 }
 m_titlefont.DeleteObject();
 m_bodyfont.DeleteObject();
}
```

```
delete []pchar;
// CView::OnPrint(pDC, pInfo);
}
```

### 举一反三

根据本实例的设计思路和一些技术要点，读者还可以：

- 设计基于对话框的打印程序。

## 实例 308

### 基于对话框结构的打印程序

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\10\308

### 实例说明

使用 Visual C++ 进行系统开发时会发现，多数的程序都是基于对话框的，在对话框中并没有封装打印功能。但是在基于对话框的程序中可以自己编写打印程序，首先把窗体的背景绘成白色，然后通过 CDC 类指针直接在窗体上绘制图像，最后根据屏幕和打印机的比例打印窗体上的图像，如图 10.2 所示。



图 10.2 基于对话框结构的打印程序

### 技术要点

通过 CPrintDialog 对话框封装的成员函数 GetPrinterDC() 可以得到打印机的设备环境。CPrintDialog 对话框封装的成员函数如表 10.1 所示。

表 10.1

CPrintDialog 对话框封装的成员函数

| 函 数 名             | 描 述                 |
|-------------------|---------------------|
| CreatePrinterDC() | 创建打印机设备环境，不显示对话框    |
| GetCopies()       | 返回打印份数              |
| DoModal()         | 显示对话框               |
| GetDefaults()     | 获取设备默认项，不显示对话框      |
| GetDevMode()      | 获取 DEVMODE 结构       |
| GetDriverName()   | 获取当前选择的打印机          |
| GetFromPage()     | 获取起始打印页数            |
| GetToPage()       | 获取终止打印页数            |
| GetPortName()     | 获取用户选择的打印机端口        |
| GetPrinterDC()    | 获取打印机 DC (设备环境)     |
| PrintAll()        | 当所有页被选中时返回 TRUE     |
| PrintCollatr()    | 当用户选择了分页复选框时返回 TRUE |
| PrintRange()      | 在设置了打印范围时返回 TRUE    |
| PrintSelection()  | 在选中某特定页时返回 TRUE     |

### 实现过程

- 新建一个基于对话框的应用程序，将窗体标题改为“基于对话框结构的打印”。
- 在窗体上添加一个按钮控件。
- 主要程序代码。

```
void CDialogPrintDlg::DrawReport(CRect rect, CDC *pDC, BOOL isprinted)
{
 titlefont.CreatePointFont(200, "宋体", pDC);
```



```
bodyfont.CreatePointFont(120,"宋体",pDC);
if (!isprinted) //预览
{
 rect.DeflateRect(0,15,0,0);
 pDC->SelectObject(&titlefont);
 pDC->DrawText("商品销售排行",rect,DT_CENTER);
 pDC->SelectObject(&bodyfont);
 CRect m_rect(rect);
 CRect temprect(m_rect.left+80,m_rect.top+60,40+(m_rect.Width())/4,m_rect.bottom+100);
 CRect itemrect;
 int width = temprect.Width();
 for (int i = 0; i < 4; i++)
 {
 pDC->DrawText(merchandise[i][0],temprect,DT_LEFT);
 itemrect.CopyRect(temprect);
 for (int y = 1; y < 5; y++)
 {
 itemrect.DeflateRect(0,50);
 pDC->DrawText(merchandise[i][y],itemrect,DT_LEFT);
 }
 temprect.DeflateRect(width,0,0,0);
 temprect.InflateRect(0,0,width,0);
 }
}
else
{
 int printx, printy;
 printx = pDC->GetDeviceCaps(LOGPIXELSX);
 printy = pDC->GetDeviceCaps(LOGPIXELSY);

 double ratex = (double)(printx)/screenx;
 double ratey = (double)(printy)/screeny;
 rect.DeflateRect(0,(int)(ratey*15),0,0);
 pDC->SelectObject(&titlefont);
 pDC->StartDoc("printinformation");
 pDC->DrawText("商品销售排行",rect,DT_CENTER);
 pDC->SelectObject(&bodyfont);
 CRect m_rect(rect);
 CRect temprect(m_rect.left+(int)(80*ratex),m_rect.top+(int)(60*ratey),
 (int)(ratey*40)+(m_rect.Width())/4,m_rect.bottom+(int)(ratey*100));
 CRect itemrect;
 int width = temprect.Width();

 for (int i = 0; i < 4; i++)
 {
 pDC->DrawText(merchandise[i][0],temprect,DT_LEFT);
 itemrect.CopyRect(temprect);
 for (int y = 1; y < 5; y++)
 {
 itemrect.DeflateRect(0,(int)(ratey*50));
 pDC->DrawText(merchandise[i][y],itemrect,DT_LEFT);
 }
 temprect.DeflateRect(width,0,0,0);
 temprect.InflateRect(0,0,width,0);
 }
 pDC->EndDoc();
}
titlefont.DeleteObject();
bodyfont.DeleteObject();
}
```

### 举一反三

根据本实例，读者可以：

- 实现各种报表和证件的打印。

## 实例 309

### 打印对话框及其控件中 的数据

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\10\309

### 实例说明

对于一些 Visual C++ 的初学者，数据打印是最难掌握的。尤其是在基于对话框的应用程序时实现打印功能。本例实现了一个简单的基于对话框应用程序的打印，能够打印对话框及其控

件中的数据。运行程序,如图 10.3 所示,单击“打印”按钮,打印效果如图 10.4 所示。

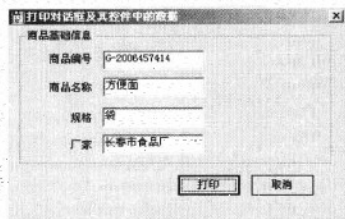


图 10.3 打印对话框及其控件中的数据图



图 10.4 打印效果

## 技术要点

要实现打印对话框,首先需要获得打印机的设备上下文 CDC 对象,其次需要将对话框界面存入位图结构中。最后调用打印机的设备上下文 CDC 对象的 StretchBlt 方法就可以了。

获取打印机的设备上下文,可以通过打印对话框实现。调用打印对话框的 GetPrinterDC 方法就可以了。例如:

```
CPrintDialog m_printdlg(FALSE);
if (m_printdlg.DoModal()==IDOK)
{
 CDC dc1;
 dc1.Attach(m_printdlg.GetPrinterDC());
}
```

将对话框界面存入位图结构中,首先定义并创建一个与对话框设备上下文兼容的 CDC 对象,本例为 memDC,然后创建一个与对话框设备上下文兼容的位图对象 bitmap,将其载入 memDC 中,最后调用 memDC 的 BitBlt 方法将对话框绘制在位图中。例如:

```
CBitmap bitmap; //定义位图对象
CClientDC dc(this); //获取对话框设备上下文
CDC memDC;
CRect rect;
//创建一个与对话框设备上下文兼容的CDC对象
memDC.CreateCompatibleDC(&dc);
this->GetClientRect(rect);
//创建位图
bitmap.CreateCompatibleBitmap(&dc,rect.Width(),rect.Height());
//载入位图
CBitmap * oldbitmap = memDC.SelectObject(&bitmap);
//将对话框绘制在位图中
memDC.BitBlt(0,0,rect.Width(),rect.Height(),&dc,0,0,SRCCOPY);
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加静态文本控件、编辑框控件和按钮控件。
- (3) 处理“打印”按钮的单击事件,打印对话框,代码如下:

```
void CPrintFormDlg::OnButton1()
{
 CBitmap bitmap;
 CClientDC dc(this);
 CDC memDC;
 CRect rect;
 memDC.CreateCompatibleDC(&dc);
 this->GetClientRect(rect);

 bitmap.CreateCompatibleBitmap(&dc,rect.Width(),rect.Height());
 CBitmap * oldbitmap = memDC.SelectObject(&bitmap);

 memDC.BitBlt(0,0,rect.Width(),rect.Height(),&dc,0,0,SRCCOPY);

 //获取打印机DC
 CPrintDialog m_printdlg(FALSE);
 if (m_printdlg.DoModal()==IDOK)
 {
 CDC dc1;
```

```
dc1.Attach(m_printdlg.GetPrinterDC());

int screenx,screeny;
int printx,printy;
double ratex,ratey;

//确定打印机与屏幕的像素比率

screenx = dc.GetDeviceCaps(LOGPIXELSX);
screeny = dc.GetDeviceCaps(LOGPIXELSY);

printx = dc1.GetDeviceCaps(LOGPIXELSX);
printy = dc1.GetDeviceCaps(LOGPIXELSY);

ratex = (double)(printx)/screenx;
ratey = (double)(printy)/screeny;

//开始打印
dc1.StartDoc("FirstDoc");
dc1.StretchBlt(0,0,(int)(rect.Width()*ratex),(int)(rect.Height()*ratey),
&memDC,0,0,rect.Width(),rect.Height(),SRCCOPY);
dc1.EndDoc();
}
memDC.SelectObject(oldbitmap);
bitmap.Detach();
}
```

### 举一反三

根据本实例，读者可以：

- 设计位图打印程序。

## 10.2 打印图片

在企业的管理系统中，经常会打印图片，本节将通过几个实例来介绍如何打印图片。

### 实例 310 打印图片

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\10\310

#### 实例说明

在开发企业人事管理系统时，经常涉及图片信息的打印。本例实现了图片信息的打印。效果如图 10.5 所示。

#### 技术要点

在 Windows 系统中，打印机的输出与屏幕的显示都是通过设备上下文 CDC 实现的。CDC 提供了 StretchBlt 方法和 BitBlt 方法用于绘制位图。在涉及图像打印时，一定要使用 StretchBlt 方法，因为该方法会根据源设备区域和目标设备区域的不同自动调整绘图的比率。而打印机的分辨率与屏幕的分辨率通常是不同的，因此需要使用 StretchBlt 方法调整打印比率。



图 10.5 打印图片

#### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加图片控件和按钮控件。从资源视图中导入一个位图，通过设置图片控件的 Image 属性将其显示在图片控件中。

(3) 处理“打印”按钮的单击事件，打印图片，代码如下：

```
void CPrintImageDlg::OnButtonprint()
{
 CDC* imagedc = m_image.GetDC();
```

```

CRect m_rect;
//获取图像的大小
m_image.GetClientRect(m_rect);

int formx,formy;
formx = imagedc->GetDeviceCaps(LOGPIXELSX);
formy = imagedc->GetDeviceCaps(LOGPIXELSY);

CPrintDialog m_printdlg (FALSE);
if (m_printdlg.DoModal()==IDOK)
{
 CDC dc;

 dc.Attach(m_printdlg.GetPrinterDC());

 //获取打印机与屏幕的分辨率比率
 int printerx,printery;
 printerx = dc.GetDeviceCaps(LOGPIXELSX);
 printery = dc.GetDeviceCaps(LOGPIXELSY);

 double ratex,ratey;
 ratex = (double)printerx/formx;
 ratey = (double)printery/formy;

 //打印图像
 dc.StartDoc("print");
 dc.StretchBlt(30,40,(int)(m_rect.Width()*ratex),(int)(m_rect.Height()*
ratey),imagedc,0,0,m_rect.Width(),m_rect.Height(),SRCCOPY);
 dc.EndDoc();
}
}

```

### 举一反三

根据本实例，读者可以：

- 设计人事报表打印程序。

## 实例 311 打印简历

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\10\311

### 实例说明

简历是许多求职者必备的资料，也是用人单位了解求职者的首选资料。它记录了求职者的学历、工作经验及特长等信息。那么如何打印简历呢？本例实现了个人简历的打印。效果如图 10.6、图 10.7 所示。

图 10.6 打印对话框

图 10.7 打印效果



## 技术要点

在打印简历时,对于简历的文字性描述可以直接从数据库中读取,但是如何读取数据库中存储的图片信息呢?图片信息在数据库中是以二进制形式进行存储的,获取图片信息可以首先获取图片占用的空间大小,然后根据其大小调用 ADO 对象的 GetChunk 方法获取,代码如下:

```
_variant_t m_bitdata;
long m_factsize = m_pRecord->GetFields()->GetItem("照片")->ActualSize;
m_bitdata = m_pRecord->GetFields()->GetItem((long)9)->GetChunk(m_factsize);
```

m\_bitdata 的数据类型为 \_variant\_t,并不是一个缓冲区指针,而我们需要获取指向位图数据的缓冲区,因此需要调用 SafeArrayAccessData 方法将一个临时缓冲区指向 m\_bitdata,然后调用 memcpy 方法将临时缓冲区指向的数据复制到一个固定的缓冲区中,代码如下:

```
static char* m_bitbuffer;
char* m_buffer = NULL;
m_bitbuffer = new char[m_factsize]; //定义一个固定的缓冲区
//将一个临时缓冲区指向m_bitdata
SafeArrayAccessData(m_bitdata.parray,(void**)&m_buffer);
//将临时缓冲区指向的数据复制到一个固定的缓冲区中
memcpy(m_bitbuffer,m_buffer,m_factsize);
```

将位图信息存入一个固定的缓冲区之后,需要调用 CreateDIBitmap 方法创建一个位图,并返回位图句柄,该方法需要知道位图信息头、位图实际数据和位图信息等。因此需要在缓冲区中获得指向这些数据的指针。为了获取位图信息头、位图实际数据和位图信息,需要了解位图的文件结构。位图由位图文件头、位图信息头、调色板和位图实际数据 4 部分组成,其中位图信息头与调色板统称为位图信息。了解了这些信息,就可以获得位图信息头、位图实际数据和位图信息了,代码如下:

```
//temp指向位图信息头
temp = m_bitbuffer+sizeof(BITMAPFILEHEADER);
BITMAPINFOHEADER* m_bitheader = (BITMAPINFOHEADER*)temp;
//获取位图信息,它包括位图信息头和调色板
BITMAPINFO* m_bitinfo = (BITMAPINFO*)temp;
//获取位图的实际数据
m_factbitdata = (void*)(m_buffer+((LPBITMAPFILEHEADER)
m_bitbuffer)->bfOffBits);
CClientDC m_dc(this);
//创建位图
hbitmap = CreateDIBitmap(m_dc.m_hDC,m_bitheader,CBM_INIT,
m_factbitdata,m_bitinfo,DIB_RGB_COLORS);
```

获得了位图句柄,然后利用 CBitmap 对象附加位图句柄,之后利用设备上下文 CDC 就可以输出位图了,代码如下:

```
m_bitmap.Attach(hbitmap);
CDC memdc;
memdc.CreateCompatibleDC(pDC);
memdc.SelectObject(&m_bitmap);
pDC->StretchBlt(410,92,100,98,&memdc,0,0,100,100,SRCCOPY);
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框类中添加成员变量,代码如下:  
CImageList m\_imagelist; //图像列表  
CReBar m\_rebar; //Rebar控件  
CToolBar m\_toolbar; //工具栏
- (3) 在对话框初始化时创建工具栏,代码如下:

```
//添加图标
for(int i = 0; i < 7; i++)
{
 m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON1+i));
}

m_rebar.Create(this,RBS_BANDBORDERS|RBS_AUTOSIZE);
m_toolbar.CreateEx(&m_rebar);
```

```

m_toolbar.GetToolBarCtrl().SetImageList(&m_imagelist);

//改变工具栏属性
m_toolbar.ModifyStyle(0, TBSTYLE_FLAT|CBRS_TOOLTIPS
|CBRS_SIZE_DYNAMIC|TBSTYLE_TRANSPARENT|TBBS_CHECKBOX);

m_toolbar.SetButtons(NULL,4);

m_toolbar.SetButtonInfo(0, IDB_PRIOR, TBSTYLE_BUTTON, 1);
m_toolbar.SetButtonText(0,"上一条");

m_toolbar.SetButtonInfo(1, IDB_NEXT, TBSTYLE_BUTTON, 2);
m_toolbar.SetButtonText(1,"下一条");

m_toolbar.SetButtonInfo(2, IDB_PRINT, TBSTYLE_BUTTON,3);
m_toolbar.SetButtonText(2,"打印");

m_toolbar.SetButtonInfo(3, IDB_QUIT, TBSTYLE_BUTTON,4);
m_toolbar.SetButtonText(3,"关闭");

m_toolbar.SetSizes(CSize(50,40),CSize(20,20));
m_rebar.AddBar(&m_toolbar);

```

(4) 处理对话框的 WM\_CTLCOLOUR 消息, 将对话框背景设置为白色, 代码如下:

```

HBRUSH CPrintResumeDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
 HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);

 CBrush brush (RGB(255,255,255));
 CRect m_rect ;
 GetClientRect(m_rect);
 pDC->FillRect(m_rect,&brush);

 return brush;
}

```

(5) 添加 DrawReport 方法绘制简历, 代码如下:

```

void CPrintResumeDlg::DrawReport(CDC* pDC,CRect m_rect,BOOL isPrinted)
{
 CString c_name,c_sex,c_age,c_knowledge,c_degree,c_phone,c_workground,
 c_suit,c_other;

 if ((! m_pRecord->ADOEOF)&&(! m_pRecord->BOF))
 {
 c_name = (TCHAR*)(_bstr_t)m_pRecord->GetFields()
->GetItem((long)1)->Value;
 c_sex = (TCHAR*)(_bstr_t)m_pRecord->GetFields()
->GetItem((long)2)->Value;
 c_age = (TCHAR*)(_bstr_t)m_pRecord->GetFields()
->GetItem((long)3)->Value;
 c_knowledge = (TCHAR*)(_bstr_t)m_pRecord->GetFields()
->GetItem((long)4)->Value;
 c_degree = (TCHAR*)(_bstr_t)m_pRecord->GetFields()
->GetItem((long)5)->Value;
 c_phone = (TCHAR*)(_bstr_t)m_pRecord->GetFields()
->GetItem((long)6)->Value;
 c_workground = (TCHAR*)(_bstr_t)m_pRecord->GetFields()
->GetItem((long)7)->Value;
 c_suit = (TCHAR*)(_bstr_t)m_pRecord->GetFields()
->GetItem((long)8)->Value;
 c_other = (TCHAR*)(_bstr_t)m_pRecord->GetFields()
->GetItem((long)10)->Value;
 }

 if (! isPrinted)
 {
 screenx = pDC->GetDeviceCaps(LOGPIXELSX);
 screeny = pDC->GetDeviceCaps(LOGPIXELSY);

 m_rect.DeflateRect(0,60,0,0);
 pDC->FillRect(&m_rect,NULL);
 pDC->DrawText("个人简历",m_rect,DT_CENTER);

 m_rect.DeflateRect(0,20);
 //绘制边框

 CRect m_framerect (m_rect);
 m_framerect.DeflateRect(10,10,10,10);
 CBrush m_brush;
 }
}

```

```

m_brush.CreateStockObject(BLACK_BRUSH);
pDC->FrameRect(&m_framerect,&m_brush);

//绘制横向区域

CRect rect(10,90,410,120);
pDC->FrameRect(&rect,&m_brush);

rect.InflateRect(0,-29,0,30);
pDC->FrameRect(&rect,&m_brush);

rect.InflateRect(0,-30,0,30);
pDC->FrameRect(&rect,&m_brush);

rect.InflateRect(0,-30,m_rect.Width()-420,30);
pDC->FrameRect(&rect,&m_brush);

rect.InflateRect(0,0,0,80);
pDC->FrameRect(&rect,&m_brush);

rect.InflateRect(0,-30,0,30);
pDC->FrameRect(&rect,&m_brush);

rect.InflateRect(0,0,0,80);
pDC->FrameRect(&rect,&m_brush);

rect.InflateRect(0,0,0,30);
pDC->FrameRect(&rect,&m_brush);

//绘制纵向区域
CRect verrect(10,90,80,180);
pDC->FrameRect(&verrect,&m_brush);

verrect.InflateRect(-69,0,130,0);
pDC->FrameRect(&verrect,&m_brush);

verrect.InflateRect(-130,0,70,0);
pDC->FrameRect(&verrect,&m_brush);

//绘制文本
/*****/
CRect textrect(10,90,80,120);
pDC->DrawText("姓名",&textrect,DT_VCENTER|DT_SINGLELINE|DT_CENTER);

textrect.InflateRect(-70,0,130,0);
pDC->DrawText(c_name,&textrect,DT_VCENTER|DT_SINGLELINE|DT_LEFT);

textrect.InflateRect(-130,0,70,0);
pDC->DrawText("学历",&textrect,DT_VCENTER|DT_SINGLELINE|DT_CENTER);

textrect.InflateRect(-70,0,130,0);
pDC->DrawText(c_knowledge,&textrect,DT_VCENTER|
DT_SINGLELINE|DT_LEFT);

/*****/
CRect rect2(10,120,80,150);
pDC->DrawText("性别",&rect2,DT_VCENTER|DT_SINGLELINE|DT_CENTER);

rect2.InflateRect(-70,0,130,0);
pDC->DrawText(c_sex,&rect2,DT_VCENTER|DT_SINGLELINE|DT_LEFT);
rect2.InflateRect(-130,0,70,0);
pDC->DrawText("电话",&rect2,DT_VCENTER|DT_SINGLELINE|DT_CENTER);

rect2.InflateRect(-70,0,130,0);
pDC->DrawText(c_phone,&rect2,DT_VCENTER|DT_SINGLELINE|DT_LEFT);

/*****/
CRect rect3(10,150,80,180);
pDC->DrawText("年龄",&rect3,DT_VCENTER|DT_SINGLELINE|DT_CENTER);

rect3.InflateRect(-70,0,130,0);
pDC->DrawText(c_age,&rect3,DT_VCENTER|DT_SINGLELINE|DT_LEFT);
rect3.InflateRect(-130,0,70,0);
pDC->DrawText("政治面貌",&rect3,DT_VCENTER|
DT_SINGLELINE|DT_CENTER);

rect3.InflateRect(-70,0,130,0);

```



```
pDC->DrawText(c_degree,&rect3,DT_VCENTER|DT_SINGLELINE|DT_LEFT);
/*****
CRect rect4(10+10,180,90,210);
pDC->DrawText("工作经验",&rect4,DT_VCENTER|DT_SINGLELINE|DT_LEFT);
rect4.InflateRect(-20,-30,m_rect.Width()-90+20,80);
pDC->DrawText(c_workground,&rect4,DT_LEFT);
/*****
CRect rect5(10+10,290,90,320);
pDC->DrawText("特长",&rect5,DT_VCENTER|DT_SINGLELINE|DT_LEFT);

rect5.InflateRect(-20,-30,m_rect.Width()-90+20,80);
pDC->DrawText(c_suit,&rect5,DT_LEFT);
/*****
CRect rect6(10+10,400,90,430);
pDC->DrawText("其他",&rect6,DT_VCENTER|DT_SINGLELINE|DT_LEFT);

rect6.InflateRect(-20,-30,m_rect.Width()-90+20,80);
pDC->DrawText(c_other,&rect6,DT_LEFT);
/*****
m_bitmap.Attach(hbitmap);
CDC memdc;
memdc.CreateCompatibleDC(pDC);
memdc.SelectObject(&m_bitmap);

pDC->StretchBlt(410,92,100,98,&memdc,0,0,100,100,SRCCOPY);
m_bitmap.Detach();
}
else
{
 printerx = pDC->GetDeviceCaps(LOGPIXELSX);
 printery = pDC->GetDeviceCaps(LOGPIXELSY);

 ratex = (double)printerx/screenx;
 ratey = (double)printery/screeny;

 m_rect.DeflateRect(10,60,0,-60);

 CClientDC dc(this);
 CDC memdc;
 CRect rect;
 memdc.CreateCompatibleDC(&dc);

 GetClientRect(rect);

 rect.DeflateRect(10,60,0,-60);

 bitmap.CreateCompatibleBitmap(&dc,rect.Width(),rect.Height());

 CBitmap* oldbitmap = memdc.SelectObject(&bitmap);

 memdc.BitBlt(0,0,rect.Width(),rect.Height(),&dc,0,0,SRCCOPY);

 rect.InflateRect(10,60,0,0);

 pDC->StartDoc("firstdoc");
 pDC->StretchBlt(0,0,(int)(ratex*(rect.Width()-10)),
(int)(ratey*(rect.Height()-60)),&memdc,10,60,rect.Width()-10,
rect.Height()-60,SRCCOPY);
 pDC->EndDoc();

 bitmap.DeleteObject();
}
}
```

(6) 添加 GetBitmapFromField 方法，从数据库中获得位图信息，代码如下：

```
HBITMAP CPrintResumeDlg::GetBitmapFromField()
{
 if ((!m_pRecord->ADOEOF)&&(!m_pRecord->BOF))
 {
 _variant t_m_bitdata;
 static char* m_bitbuffer;
 char* m_buffer = NULL;
 char* temp = NULL;
 long m_factsize = m_pRecord->GetFields()
->GetItem("照片")->ActualSize;

 //获取位图所有数据
 m_bitdata = m_pRecord->GetFields()->GetItem((long)9)
->GetChunk(m_factsize);
 HBITMAP m_hmap;
```





```

if(m_bitdata.vt==VT_ARRAY[VT_UI1])
{
 //定义一个数据缓冲区
 m_bitbuffer = new char[m_factsize];
 //将m_buffer指向m_bitdata
 SafeArrayAccessData(m_bitdata.parray,(void*)&m_buffer);
 //复制位图数据到m_bitbuffer;
 memcpy(m_bitbuffer,m_buffer,m_factsize);

 SafeArrayUnaccessData(m_bitdata.parray);

 void* m_factbitdata; //实际的位图数据

 //temp指向位图信息头
 temp = m_bitbuffer+sizeof(BITMAPFILEHEADER);

 BITMAPINFOHEADER * m_bitheader = (BITMAPINFOHEADER*)temp;

 //获取位图信息,它包括位图信息头和调色板
 BITMAPINFO* m_bitinfo = (BITMAPINFO*)temp;

 //获取位图的实际数据
 m_factbitdata = (void*)(m_buffer+((LPBITMAPFILEHEADER)
m_bitbuffer)->bfOffBits);

 CClientDC m_dc(this);
 //创建位图
 hbitmap = CreateDIBitmap(m_dc.m_hDC,m_bitheader,CBM_INIT,
m_factbitdata,m_bitinfo,DIB_RGB_COLORS);
 delete [] m_bitbuffer;
}
return hbitmap;
}

```

### 举一反三

根据本实例，读者可以：

- 设计复杂的报表打印。

## 10.3 打印单据

在大型的商场或物流公司中，各种单据已经成为日常工作中必不可少的文件，本节将通过几个实例介绍单据打印。

### 实例 312 打印汇款单

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrsoft\10\312

#### 实例说明

在开发数据库应用程序时，经常需要设计各种类型的报表，其中汇款单式的报表是比较常用的一种报表，如何设计汇款单式报表呢？本例设计了一个汇款单式报表，效果如图 10.8 所示。

#### 技术要点

设计汇款单式报表关键是绘制表格。如何绘制表格呢？在 MFC 中似乎没有现成的控件可以使用。可以从 CStatic 派生一个子类，在该类中实现一个矩形的绘制，并提供方法可以控制边框的颜色以及每个边框的显示。

首先从 CStatic 派生一个子类，本例为 CMyStatic。在该类中定义成员变量，控制控件的外观，代码如下：

图 10.8 打印汇款单

```

BOOL islined; //是否有下划线
BOOL isframed; //是否有边框

BOOL isLeft; //是否有左边线
BOOL isRight; //是否有右边线
BOOL isTop; //是否有上边线
BOOL isBottom; //是否有下边线

COLORREF m_color; //文本颜色
COLORREF m_framecolor; //边框颜色
UINT m_lindwidth; //边线宽度
UINT align; //文本对齐方式

```

然后处理 CMystic 的 WM\_PAINT 消息，绘制边框及文本，代码如下：

```

void CMystic::OnPaint()
{
 CPaintDC dc1(this);

 CDC dc;
 dc.Attach(GetDC()->m_hDC);

 CRect rect;
 GetClientRect(rect);
 dc.SetBkMode(TRANSPARENT);

 CPen pen(PS_SOLID,m_lindwidth,m_framecolor);

 CPen* oldpen;
 oldpen = dc.SelectObject(&pen);

 if (isframed)
 {
 if (isTop)
 {
 dc.MoveTo(rect.left,rect.top); //上边线
 dc.LineTo(rect.right,rect.top);
 }

 if (isBottom)
 {
 dc.MoveTo(rect.left,rect.bottom) //下边线
 dc.LineTo(rect.right,rect.bottom);
 }

 if (isLeft)
 {
 dc.MoveTo(rect.left,rect.top); //左边线
 dc.LineTo(rect.left,rect.bottom);
 }

 if (isRight)
 }
}

```

```

 {
 dc.MoveTo(rect.right,rect.top); //右边线
 dc.LineTo(rect.right,rect.bottom);
 }

 if (islined)
 {
 CPen linepen(PS_SOLID,m_lindwidth,m_color);
 oldpen = dc.SelectObject(&linepen);

 dc.MoveTo(rect.left,rect.bottom-1);
 dc.LineTo(rect.right,rect.bottom-1);
 }

 dc.SelectObject(oldpen);
 CString str ;
 GetWindowText(str);
 dc.SetTextColor(m_color);
 //绘制文本
 dc.DrawText(str,rect,align);
}

```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加静态文本和按钮控件。在类向导中为控件命名,如图 10.9 所示。

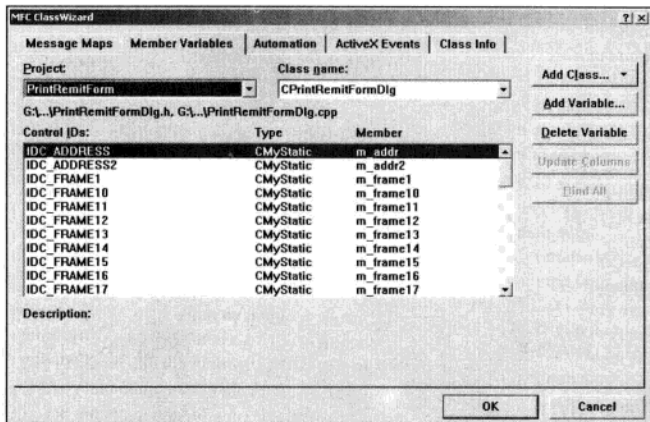


图 10.9 类向导

- (3) 向对话框类中添加 OnPrint 方法,实现打印功能,代码如下:

```

void CPrintRemitFormDlg::OnPrint()
{
 CWnd* m_btn = GetDlgItem(IDC_PRINT);
 m_btn->ShowWindow(SW_HIDE);

 CBitmap bitmap;
 CClientDC dc(this);
 CDC memDC;
 CRect rect;
 memDC.CreateCompatibleDC(&dc);
 this->GetClientRect(rect);

 bitmap.CreateCompatibleBitmap(&dc,rect.Width(),rect.Height());
 CBitmap * oldbitmap = memDC.SelectObject(&bitmap);

 memDC.BitBlt(0,0,rect.Width(),rect.Height(),&dc,0,0,SRCCOPY);

 //获取打印机DC
 CPrintDialog m_printdlg(FALSE);
 if (m_printdlg.DoModal()==IDOK)
 {
 CDC dc1;
 dc1.Attach(m_printdlg.GetPrinterDC());
 }
}

```

```
int screenx,screeny;
int printx,printy;
double ratex,ratey;

//确定打印机与屏幕的像素比率
screenx = dc.GetDeviceCaps(LOGPIXELSX);
screeny = dc.GetDeviceCaps(LOGPIXELSY);

printx = dc1.GetDeviceCaps(LOGPIXELSX);
printy = dc1.GetDeviceCaps(LOGPIXELSY);

ratex = (double)(printx)/screenx;
ratey = (double)(printy)/screeny;

//开始打印
dc1.StartDoc("FirstDoc");
dc1.StretchBlt(0,0,(int)(rect.Width()*ratex),(int)(rect.Height()*ratey)
,&memDC,0,0,rect.Width(),rect.Height(),SRCCOPY);
dc1.EndDoc();

memDC.SelectObject(oldbitmap);
bitmap.Detach();
}
m_btn->ShowWindow(SW_SHOW);
}
```

### 举一反三

根据本实例的设计思路和一些技术要点，读者还可以：

- 自定义编辑框控件。

## 实例 313 打印信封标签

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\10\313

### 实例说明

信件在生活中有着传递信息的重要作用。当处理大量的商业信息时，人工填写信封是非常繁琐的一件事，这时使用计算机打印信封就变得十分的重要了。运行程序，选择要打印的数据，然后在编辑框中输入寄信人的邮编，单击“打印”按钮进行打印，也可以单击“打印预览”按钮进行预览，如图 10.10 所示。

### 技术要点

信封的打印关键是数据的位置，只要把收信人的地址、姓名、邮编等信息准确地打印在信封的固定位置上就可以了。通过基于对话框的程序打印信封，直接在窗体上显示预览图像，然后调用 CPrintDialog 打印对话框完成打印。

### 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“打印信封标签”。
- (2) 向窗体中添加一个编辑框控件、一个列表视图控件和两个按钮控件。
- (3) 主要程序代码。

```
void CPreview::DrawReport(CRect rect, CDC *pDC, BOOL isprinted)
{
 titlefont.CreatePointFont(110, _T("宋体"), pDC);
 int printx, printy;
 printx = pDC->GetDeviceCaps(LOGPIXELSX);
```

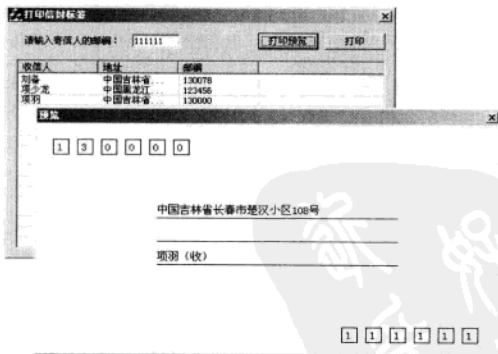


图 10.10 打印信封标签



```

printy = pDC->GetDeviceCaps(LOGPIXELSY);
double ratex = (double)(printx)/screenx;
double ratey = (double)(printy)/screeny;
if(isprinted)
{
 pDC->StartDoc("printinformation");
}
else
{
 ratex=1,ratey=1;
}
for(int i=0;i<6;i++)
{
 pDC->SelectObject(&titlefont);
 pDC->Rectangle((int)((20+i*30)*ratex),(int)(20*ratey),(int)((40+i*30)*ratex),(int)(40*ratey));
 if(i==0)
 {
 rect.DeflateRect((int)(25*ratex),(int)(23*ratey),0,0);
 pDC->DrawText(sarrays[0][i],rect,DT_LEFT);
 }
 else
 {
 rect.DeflateRect((int)(-330*ratex),(int)(-240*ratey),0,0);
 pDC->DrawText(sarrays[0][i],rect,DT_LEFT);
 }
 pDC->Rectangle((int)((380+i*30)*ratex),(int)(260*ratey),(int)((400+i*30)*ratex),(int)(280*ratey));
 rect.DeflateRect((int)(360*ratex),(int)(240*ratey),0,0);
 pDC->DrawText(sarrays[1][i],rect,DT_LEFT);
}
pDC->MoveTo((int)(150*ratex),(int)(120*ratey));
pDC->LineTo((int)(450*ratex),(int)(120*ratey));
pDC->MoveTo((int)(150*ratex),(int)(150*ratey));
pDC->LineTo((int)(450*ratex),(int)(150*ratey));
pDC->MoveTo((int)(150*ratex),(int)(180*ratey));
pDC->LineTo((int)(450*ratex),(int)(180*ratey));
CString str1,str2;
int n;
n = strText.GetLength();
if(n/3<42)
{
 str1=strText.Left(42);
 rect.DeflateRect((int)(-385*ratex),(int)(-160*ratey),0,0);
 pDC->DrawText(str1,rect,DT_LEFT);
 str2=strText.Right(n-42);
 rect.DeflateRect(0,(int)(30*ratey),0,0);
 pDC->DrawText(str2,rect,DT_LEFT);
 rect.DeflateRect(0,(int)(30*ratey),0,0);
 pDC->DrawText(strsxr,rect,DT_LEFT);
}
else
{
 str1=strText.Left(n/2);
 rect.DeflateRect((int)(-385*ratex),(int)(-160*ratey),0,0);
 pDC->DrawText(str1,rect,DT_LEFT);
 str2=strText.Right(n-n/2);
 rect.DeflateRect(0,(int)(30*ratey),0,0);
 pDC->DrawText(str2,rect,DT_LEFT);
 rect.DeflateRect(0,(int)(30*ratey),0,0);
 pDC->DrawText(strsxr,rect,DT_LEFT);
}
if(isprinted)
{
 pDC->EndDoc();
}
titlefont.DeleteObject();
}

```

## 实例 314 假条套打

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\10\314

## 实例说明

套打打印与普通的打印是不同的，套打打印的形式一般采用自定义大小的纸张，所以属于不规格的纸张大小。如图 10.11 所示。

图 10.11 假条套打

## 技术要点

对于一般的报表或表格的打印都是以打印机画布的左上角的某一坐标作为打印的原点的，根据这一点和报表或表格的自身大小来绘制。也就是说打印的内容由报表或表格自身的设计而定，不受外界的任何影响。而套打打印却与一般的打印不同，套打打印的打印内容是需要与已存在的打印纸的大小和内容所在的位置一致的。不能出现太大的偏差，否则数据就会出现移位的情况。

其实，套打打印的设计方法只有两种：一种是为每个需要打印的内容设计坐标，用户可以通过调整坐标中的值来完成打印内容的准确性；另一种方法则是根据已存在打印纸及内容的位置来设套打打印程序。当然，第二种方法要比第一种方法简单并容易实现的多，但修改起来就不那么容易了，该实例使用的是第二种方法。

在进行套打打印时需要对面体中所有控件进行循环获取，并判断该控件的类型。这时可以使用 GetDlgItem 方法通过代表窗体中控件的标识符来获取控件，但此时控件的类型为 CWnd 窗口类型，所以要想得到更准确的类型就必需使用 GetClassName 函数来获取类名称来判断。实现代码如下：

```
char classname[255];
char * se = "Edit"; //编辑框控件类型名
char * sb = "Button"; //按钮类控件类型名
CRect rect;
for (int i = 1002; i <= 1031; i++) //循环控件标识符
{
 CWnd *control = this->GetDlgItem(i); //获取控件
 if (NULL != control)
 {
 GetClassName(control->GetSafeHwnd(), classname, 255); //获取类名
 if (strcmp(classname, se) == 0) //判断类型
 {
 CString str;
 control->GetWindowText(str);
 control->GetWindowRect(&rect); //获取位置
 pdc.TextOut(rect.left * rateX, rect.left * rateY, str); //输出内容
 } else if (strcmp(classname, sb) == 0)
 {
 if (((CButton *)control)->GetCheck() == 1)
 {
 control->GetWindowRect(&rect);
 pdc.TextOut(rect.left * rateX, rect.left * rateY, "√");
 }
 }
 }
}
```

通过上段代码可以看出循环中的变量  $i$  是 1 002~1 031 之间的整数值,也就代表了窗体中控件标识符的范围,这个值又是怎么得来的呢?其实当程序员在窗体上添加控件时,系统会自动为其设置一个标识符,而这些标识符都被定义在 Resource.h 这个头文件中,只要打开该头文件便可知道标识符的范围了。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 向窗体中添加若干个文本编辑框控件、若干个静态文本控件、若干个复选框控件和两个按钮控件等,并按照假条的样式及大小设计窗体。

(3) 主要程序代码。

```
void CPrintLeaveDlg::OnPrint()
{
 CClientDC sdc(this);
 CPrintDialog pdlg(false); //定义打印机对话框
 pdlg.GetDefaults(); //获取打印机默认值
 CDC pdc;
 pdc.Attach(pdlg.GetPrinterDC()); //关联打印机

 double ratex = pdc.GetDeviceCaps(LOGPIXELSX) /
 sdc.GetDeviceCaps(LOGPIXELSX); //屏幕与打印机分辨率比率(水平方向)
 double ratey = pdc.GetDeviceCaps(LOGPIXELSY) /
 sdc.GetDeviceCaps(LOGPIXELSY); //屏幕与打印机分辨率比率(垂直方向)

 pdc.StartDoc("print"); //开始打印
 char classname[255];
 char * se = "Edit"; //编辑框控件类型名
 char * sb = "Button"; //按钮类控件类型名
 CRect rect;
 for (int i = 1002; i <= 1031; i++) //循环控件标识符
 {
 CWnd *control = this->GetDlgItem(i); //获取控件
 if (NULL != control)
 {
 GetClassName(control->GetSafeHwnd(), classname, 255); //获取类名
 if (strcmp(classname, se) == 0) //判断类型
 {
 CString str;
 control->GetWindowText(str);
 control->GetWindowRect(&rect); //获取位置
 pdc.TextOut(rect.left * ratex, rect.left * ratey, str); //输出内容
 }
 else if (strcmp(classname, sb) == 0)
 {
 if (((CButton *)control)->GetCheck() == 1)
 {
 control->GetWindowRect(&rect);
 pdc.TextOut(rect.left * ratex, rect.left * ratey, "√");
 }
 }
 }
 }
 pdc.EndDoc(); //结束打印
}
```

## 举一反三

根据本实例,读者可以:

- 实现收据打印。

## 实例 315 批量打印条形码

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\10\315

## 实例说明

在商业管理中,为了管理方便,经常要对某些商品设置条形码。但逐个打印条形码费时费

力, 本实例将实现批量打印条形码的功能。运行程序, 如图 10.12 所示, 单击“打印”按钮, 将批量打印数据库中条形码, 打印效果如图 10.13 所示。



图 10.12 批量打印条形码



图 10.13 打印效果

## 技术要点

本例使用 GetDIBits 函数和 StretchDIBits 函数将条形码控件的客户区域保存成位图并打印。

GetDIBits 函数将来自一幅位图的二进制位复制到一幅与设备无关的位图里, 语法如下:

```
int GetDIBits(HDC hdc, HBITMAP hbm, UINT uStartScan, UINT cScanLines, LPVOID lpvBits, LPBITMAPINFO lpbi, UINT uUsage);
```

参数说明:

- hdc: 定义了与设备有关位图 hBitmap 的配置信息的一个设备场景的句柄。
- hbm: 源位图的句柄。
- uStartScan: 要复制到 DIB 中的第一条扫描线的编号。
- cScanLines: 要复制的扫描线数量。
- lpvBits: 指向一个缓冲区的指针。
- lpbi: 对 lpvBits 的格式及颜色进行说明的一个结构。
- uUsage: 为 DIB\_PAL\_COLORS 和 DIB\_RGB\_COLORS 常数之一, 其中 DIB\_PAL\_COLORS 表示在颜色表中装载一个 16 位索引值数组, 它们与当前选定的调色板有关; DIB\_RGB\_COLORS 表示在颜色表中装载 RGB 颜色。

StretchDIBits 函数将一幅与设备无关位图的全部或部分数据直接复制到指定的设备场景,

语法如下:

```
int StretchDIBits(HDC hdc, int XDest, int YDest, int nDestWidth, int nDestHeight, int XSrc, int YSrc, int nSrcWidth, int nSrcHeight, CONST VOID *lpBits, CONST BITMAPINFO *lpBitsInfo, UINT iUsage, DWORD dwRop);
```

参数说明:

- hdc: 一个设备场景的句柄。该场景用于接收位图数据。
- Xdest: 用逻辑坐标表示的目标矩形的起点横坐标。
- Ydest: 用逻辑坐标表示的目标矩形的起点纵坐标。
- NDestWidth: 目标矩形的宽度。
- nDestHeight: 目标矩形的高度。
- XSrc: 用设备坐标表示的源矩形在 DIB 中的起点横坐标。
- Ysrc: 用设备坐标表示的源矩形在 DIB 中的起点纵坐标。
- NSrcWidth: 源矩形的宽度。
- nSrcHeight: 源矩形的高度。
- lpBits: 指向用来检索位图数据的缓冲区指针, 如果此参数为 NULL, 那么函数将把位



图的维数与格式传递给 lpbi 参数指向的 BITMAPINFO 结构。

- lpBitsInfo: 对 lpBits 的格式和颜色进行描述的一个结构。
- iUsage: 常数, DIB\_PAL\_COLORS 和 DIB\_RGB\_COLORS 两个值之一, 其中 DIB\_PAL\_COLORS 表示颜色表是一个整数数组, 其中包含了与目前选入 hdc 设备场景的调色板相关的索引; DIB\_RGB\_COLORS 表示颜色表包含了 RGB 颜色。
- dwRop: 光栅操作类型。

## 实现过程

(1) 新建一个基于对话框的应用程序, 将窗体标题改为批量打印条形码。

(2) 选择菜单 Project/Add To Project, 然后选择 Components and Controls..., 打开 Components and Controls Gallery 窗口, 双击窗口中的 Registered ActiveX Controls 文件夹, 找到 Microsoft BarCode Control 9.0 选项, 双击它取默认值, 添加控件, 单击“Close”按钮。BarCode 控件就添加到控件面板中了。

(3) 向窗体中添加一个 BarCode 控件、一个列表视图控件和两个按钮控件。

(4) 主要程序代码。

```
void CPrintBarcodeDlg::OnPrint()
{
 CString str;
 //构造打印对话框
 CPrintDialog m_print(false, PD_ALLPAGES | PD_USEDEVMODECOPIES | PD_RETURNDEFAULT | PD_NOPAGENUMS |
 PD_HIDEPRINTTOFILE | PD_NOSELECTION, this);
 m_Barcode.SetBackColor(RGB(255,255,255));
 if (m_print.DoModal() != IDOK)
 {
 CDC dc1;
 dc1.Attach(m_print.GetPrinterDC()); //将dc1关联到打印机句柄
 dc1.StartDoc("print");

 int screenx, screeny;
 int printx, printy;
 float ratex, ratey;
 //确定打印机与屏幕的像素比率
 screenx = m_Barcode.GetDC()->GetDeviceCaps(LOGPIXELSX);
 screeny = m_Barcode.GetDC()->GetDeviceCaps(LOGPIXELSY);
 printx = dc1.GetDeviceCaps(LOGPIXELSX);
 printy = dc1.GetDeviceCaps(LOGPIXELSY);
 ratex = (float)(printx)/screenx;
 ratey = (float)(printy)/screeny;

 for (int row = 0; row < m_List.GetItemCount(); row++)
 {
 str = m_List.GetItemText(row, 0);
 m_Barcode.SetValue((COleVariant)str);
 m_Barcode.UpdateWindow();
 CDC* pBar = m_Barcode.GetDC();
 pBar->SetBkMode(TRANSPARENT);
 CDC memdc;
 memdc.CreateCompatibleDC(pBar);
 CRect rect;
 m_Barcode.GetClientRect(rect); //获取条形码的客户区域
 CBitmap bitmap;
 bitmap.CreateCompatibleBitmap(pBar, rect.Width(), rect.Height());
 memdc.SelectObject(&bitmap);
 memdc.BitBlt(0, 0, rect.Width(), rect.Height(), pBar, 0, 0, SRCCOPY);

 BITMAP bmp;
 bitmap.GetBitmap(&bmp);
 int panelsize = 0; //调色版大小
 if (bmp.bmBitsPixel < 16)
 panelsize = pow(2, bmp.bmBitsPixel) * sizeof(RGBQUAD);
 BITMAPINFO* blnfo = (BITMAPINFO*)LocalAlloc(LPTR, sizeof(BITMAPINFO) + panelsize);
 blnfo->bmiHeader.biBitCount = bmp.bmBitsPixel;
 blnfo->bmiHeader.biClrImportant = 0;
 blnfo->bmiHeader.biClrUsed = 0;
 blnfo->bmiHeader.biCompression = 0;
 blnfo->bmiHeader.biHeight = bmp.bmHeight;
 blnfo->bmiHeader.biPlanes = bmp.bmPlanes;
 blnfo->bmiHeader.biSize = sizeof(BITMAPINFO);
```

```

bInfo->bmiHeader.biSizeImage = bmp.bmWidthBytes*bmp.bmHeight;
bInfo->bmiHeader.biWidth = bmp.bmWidth;
bInfo->bmiHeader.biXPelsPerMeter = 0;
bInfo->bmiHeader.biYPelsPerMeter = 0;

```

```

char* pdata = new char[bmp.bmWidthBytes*bmp.bmHeight];
GetDIBits(memdc.m_hDC,bitmap,0,bmp.bmHeight,pdata,bInfo,DIB_RGB_COLORS);

```

```

CDC* pDC = GetDC();

```

```

//开始打印

```

```

int x = bInfo->bmiHeader.biWidth;

```

```

int y = bInfo->bmiHeader.biHeight;

```

```

StretchDIBits(dc1.m_hDC,80,row*

```

```

(int)(200*rateX),int(x*rateX),int(y*rateY),1,1,x,y,pdata,bInfo,DIB_RGB_COLORS,SRCCOPY);

```

```

delete pdata;

```

```

LocalFree(bInfo);

```

```

}
dc1.EndDoc();

```

### 举一反三

根据本实例，读者可以：

● 打印其他图表。

## 10.4 控制打印

编写打印程序之所以复杂，是因为其处理的信息比较复杂，所以要想编写一个好的打印程序，对打印相关的控制技术的掌握是必不可少的。本节通过几个实例对控制打印技术进行简单的介绍。

### 实例 316 批量打印文档

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\10\316

### 实例说明

在一些企事业单位中，有时需要同时打印多篇 Word 文档，如果手工逐篇打开文档，再进行打印，则费时费力。如果通过程序控制，在不打开文档的情况下进行批量打印 Word 文档，则将会节省大量的时间。本实例将实现批量打印 Word 文档的功能。运行程序，单击“选择文件夹”按钮选择指定文件夹，将该文件夹下的文件显示在列表中，如图 10.14 所示，单击“打印”即可打印该文件夹内的所有 Word 文档。



图 10.14 批量打印文档

### 技术要点

在 For 循环语句中应用 ShellExecute 函数调用 Windows 的“print”打印命令，从而实现批量打印 Word 文档的功能。

### 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为批量打印文档。
- (2) 向窗体中添加一个 ListBox 控件和两个按钮控件。
- (3) 主要程序代码。

```

void CPrintWordsDlg::OnButliulan()

```

```

{
// TODO: Add your control notification handler code here
CString ReturnPach;
TCHAR szPath[_MAX_PATH];

```

```
BROWSEINFO bi;
bi.hwndOwner=NULL;
bi.pidlRoot=NULL;
bi.lpszTitle=_T("请选择一个文件夹");
bi.pszDisplayName=szPath;
bi.ulFlags=BIF_RETURNONLYFSDIRS;
bi.lpfm=NULL;
bi.lParam=NULL;
LPITEMIDLIST pltemIDList=SHBrowseForFolder(&bi);
if(pltemIDList)
{
 if(SHGetPathFromIDList(pltemIDList,szPath))
 ReturnPach=szPath;
}
else
 ReturnPach="";

CFileFind file;
if(ReturnPach.Right(1)!="\\")
 ReturnPach += "*.doc";
BOOL bf;
bf = file.FindFile(ReturnPach);
int i=0;
while(bf)
{
 bf = file.FindNextFile();
 if (!file.IsDots())
 {
 StrPath[i] = file.GetFilePath();
 StrName = file.GetFileName();
 m_List.AddString(StrName);
 i++;
 }
}
}

void CPrintWordsDlg::OnButprint()
{
 // TODO: Add your control notification handler code here
 for(int i=0;i<m_List.GetCount();i++)
 {
 ::ShellExecute(NULL,"print",StrPath[i],"",SW_HIDE);
 }
}
```

### 举一反三

根据本实例，读者可以：

- 批量打印 Excel 文档。

## 实例 317 实现横向打印

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\10\317

### 实例说明

在设计报表打印时，许多开发人员都会遇到这样一个问题，待打印的数据超出了打印纸的宽度，导致在打印时只有部分数据被打印出来。通常可以有两种方式解决，一种是折行打印，另一种是横向打印，本例中笔者采用第二种方式解决。运行程序，如图 10.15 所示。

### 技术要点

通过 GetDevMode 函数获取 DEVMODE 结构，该结构记录了打印机的所有信息，本例实现横向打



图 10.15 实现横向打印

印, 需要设置 DEVMODE 结构的 dmOrientation 字段为 DMORIENT\_LANDSCAPE。

## 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为实现横向打印。
- (2) 向窗体中添加一个按钮控件。
- (3) 主要程序代码。

```
void CPrintBearingDlg::OnButPrint()
{
 // TODO: Add your control notification handler code here
 DWORD dwflags=PD_ALLPAGES | PD_NOPAGENUMS | PD_USEDEVMODECOPIES
 | PD_SELECTION | PD_HIDEPRINTTOFILE;
 CPrintDialog dlg(false,dwflags,NULL);
 if(dlg.DoModal() == IDOK)
 {
 LPDEVMODE dev = dlg.GetDevMode();
 dev->dmFields = DM_PAPERSIZE | DM_PAPERWIDTH
 | DM_PAPERLENGTH | dev->dmFields;
 dev->dmFields = dev->dmFields | DMBIN_MANUAL;
 dev->dmDefaultSource = DMBIN_MANUAL;
 dev->dmOrientation = DMORIENT_LANDSCAPE;
 char aa[32] = "自定义";
 strcpy((char*)dev->dmFormName,aa);

 CDC dc;
 dc.Attach(dlg.CreatePrinterDC());
 CDC* pDC = &dc;
 int leftmargin;
 leftmargin = dc.GetDeviceCaps(PHYSICALOFFSETX);
 CRect rect
(-leftmargin,0,dc.GetDeviceCaps(PHYSICALWIDTH)-leftmargin,dc.GetDeviceCaps(PHYSICALHEIGHT));
 titlefont.CreatePointFont(200,"宋体",pDC);
 bodyfont.CreatePointFont(120,"宋体",pDC);
 int printx,printy;
 printx = pDC->GetDeviceCaps(LOGPIXELSX);
 printy = pDC->GetDeviceCaps(LOGPIXELSY);

 double ratex = (double)(printx)/screenx;
 double ratey = (double)(printy)/screeny;
 rect.DeflateRect(0,(int)(ratey*15),0,0);
 pDC->SelectObject(&titlefont);
 pDC->StartDoc("print");
 pDC->DrawText("商品销售排行",rect,DT_CENTER);
 pDC->SelectObject(&bodyfont);
 CRect m_rect(rect);
 CRect temprect(m_rect.left+(int)(80*ratex),m_rect.top+(int)(60*ratey),
 (int)(ratey*40)+(m_rect.Width()/4,m_rect.bottom+(int)(ratey*100));
 CRect itemrect;
 int width = temprect.Width();

 for (int i = 0;i<4;i++)
 {
 pDC->DrawText(merchandise[i][0],temprect,DT_LEFT);
 itemrect.CopyRect(temprect);
 for (int y = 1; y< 5;y++)
 {
 itemrect.DeflateRect(0,(int)(ratey*50));
 pDC->DrawText(merchandise[i][y],itemrect,DT_LEFT);
 }
 temprect.DeflateRect(width,0,0,0);
 temprect.InflateRect(0,0,width,0);
 }
 titlefont.DeleteObject();
 bodyfont.DeleteObject();
 }
}
```

## 举一反三

根据本实例, 读者可以:

- 在程序中随意控制打印机信息。



## 实例 318

设置打印表格的边线及  
字体

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\10\318

## 实例说明

表格的打印是报表打印设计中常见的一种形式,所以在对表格进行打印时边线及字体的设置也是非常重要的,如图 10.16 所示。

## 技术要点

对于表格的绘制是非常简单的了,只要循环表格的所有行和列来绘制组成表格的线条,再将表格中的文字绘制到相应的单元格中,即可完成对表格的绘制操作。对于线条的宽度就需要通过指定的设置来完成了,而字体的选择也是需要通过指定的设置并记录该字体的 LOGFONT 结构信息。记录字体的结构信息代码如下:

```
void CPrintLineandFontDlg::OnSelectFont()
{
 CFontDialog fontdlg;
 if (fontdlg.DoModal() == IDOK)
 {
 fontdlg.GetCurrentFont(&dlg.logfont)
 dlg.isfont = true;
 }
}
```

//定义字体对话框  
//选择成功  
//获取结构信息;  
//使用新字体标记

在绘制表格中的文字时就可以通过选择的字体进行绘制了,实现代码如下:

```
CFont font;
if (isfont)
 font.CreateFontIndirect(&logfont);
else
 font.CreatePointFont(90,"宋体",dc);
dc->SelectObject(&font);
```

//定义字体  
//使用新字体  
//通过结构信息创建字体  
//使用默认字体  
//选择字体

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向窗体中添加 1 个编辑框控件、1 个静态文本控件、1 个列表视图控件和 3 个按钮控件。
- (3) 插入一个新的对话框资源,设置关联类名为 CPreview。
- (4) 在打印表格时应先将数据写入列表视图控件,而这一操作应该添加在窗体的 OnInitDialog 方法中,实现代码如下:

```
m_Grid.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_FULLROWSELECT|LVS_EX_GRIDLINES); //设置表格类型
m_Grid.InsertColumn(0,"姓名",LVCFMT_LEFT,100); //添加列
m_Grid.InsertColumn(1,"性别",LVCFMT_LEFT,100);
m_Grid.InsertColumn(2,"年龄",LVCFMT_LEFT,100);
m_Grid.InsertColumn(3,"地址",LVCFMT_LEFT,100);

m_Grid.InsertItem(0,"张三"); //添加行
m_Grid.SetItemText(0,1,"男");
m_Grid.SetItemText(0,2,"18");
m_Grid.SetItemText(0,3,"吉林");

m_Grid.InsertItem(1,"李四");
m_Grid.SetItemText(1,1,"男");
m_Grid.SetItemText(1,2,"25");
m_Grid.SetItemText(1,3,"长沙");

m_Grid.InsertItem(2,"王五");
m_Grid.SetItemText(2,1,"男");
m_Grid.SetItemText(2,2,"30");
```

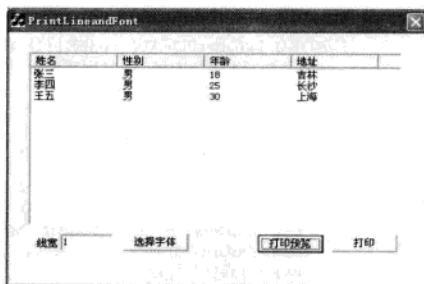


图 10.16 设置所打印表格的边线及字体

```
m_Grid.SetItemText(2,3,"上海");
```

```
dlg.PrintGrid = &m_Grid;
dlg.isfont = false;
m_edit = "1";
UpdateData(false);
```

```
//使用默认字体
//线宽默认为1
//更新数据
```

(5) GetPrintRate 方法是预览类 CPreview 用来计算打印机与屏幕分辨率比率的,实现代码如下:

```
void CPreview::GetPrintRate(double &ratex,double &ratey)
```

```
{
 CClientDC sdc(this);
 CPrintDialog pdlg(false);
 pdlg.GetDefaults();
 CDC pdc;
 pdc.Attach(pdlg.GetPrinterDC());
 ratex = pdc.GetDeviceCaps(LOGPIXELSX) /
 sdc.GetDeviceCaps(LOGPIXELSX);
 ratey = pdc.GetDeviceCaps(LOGPIXELSY) /
 sdc.GetDeviceCaps(LOGPIXELSY);
}
```

(6) PreviewGrid 方法则是 CPreview 类中的主要方法,该方法不但实现了表格的预览绘制,还实现了表格的打印绘制。并根据线宽与字体的设置绘制表格,实现代码如下:

```
void CPreview::PreviewGrid(CDC * dc,bool isprint)
```

```
{
 if (PrintGrid == NULL) //打印表格为空退出
 return ;

 double ratex,ratey;
 if (isprint)
 GetPrintRate(ratex,ratey); //计算屏幕与打印机分辨率比率
 else
 ratex = ratey = 1; //预览时比率为1

 int columns = PrintGrid->GetHeaderCtrl()->GetItemCount(); //表格列数
 int items = PrintGrid->GetItemCount(); //表格行数
 CFont font;
 if (isfont)
 font.CreateFontIndirect(&logfont); //创建指定字体
 else
 font.CreatePointFont(90,"宋体",dc); //创建默认字体
 dc->SelectObject(&font); //选择字体
 CPen pen(PS_SOLID,LineWidth,RGB(0,0,0)); //创建指定线宽画笔
 if (LineWidth > 0)
 dc->SelectObject(&pen); //选择画笔
 for (int i = 0;i < columns;i++) //循环列绘制表格头
 {
 CRect rect;
 PrintGrid->GetHeaderCtrl()->GetItemRect(i,&rect);
 rect.left *= ratex;
 rect.top *= ratey;
 rect.right *= ratex;
 rect.bottom *= ratey;
 if (i>0)
 rect.OffsetRect(-(int)(i*ratex),0);
 dc->Rectangle(&rect);
 LV_COLUMN column;
 char mm[255];
 column.mask = LVCF_TEXT;
 column.pszText = mm;
 column.cchTextMax = 255;
 PrintGrid->GetColumn(i,&column);
 dc->SetBkMode(TRANSPARENT);
 dc->TextOut(rect.left + 2*ratex,rect.top + 2*ratey,column.pszText);
 }
 for (int n = 0 ;n < items;n++) //循环行与列绘制表格体及文字
 {
 CRect rect;
 PrintGrid->GetItemRect(n,&rect,LVIR_BOUNDS);
 rect.left *= ratex;
 rect.top *= ratey;
 rect.right *= ratex;
 rect.bottom *= ratey;

 rect.OffsetRect(0,-(3+n)*ratey);
 rect.InflateRect(0,0,-(columns - 1)*ratex,0);
 dc->Rectangle(&rect);
 for (int m = 0;m < columns;m++)
 {
```

```

CRect rectw;
PrintGrid->GetHeaderCtrl()->GetItemRect(m,&rectw);
rectw.left *= ratex;
rectw.top *= ratey;
rectw.right *= ratex;
rectw.bottom *= ratey;
dc->MoveTo(rectw.right - m - 1, rectw.top);
dc->LineTo(rectw.right - m - 1, rectw.bottom);
CString text = PrintGrid->GetItemText(n,m);
dc->TextOut(rectw.left, rectw.top, text);
}
}

```

### 举一反三

根据本实例，读者可以：

- 绘制表格式报表。

## 10.5 打印预览

如今，用户在挑选软件时，不仅对实用性有要求，同时对界面的要求也不断提高，本节将通过几个实例来介绍如何修改打印预览界面。

### 实例 319 具有滚动条的预览界面

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\10\319

#### 实例说明

用户在对程序中的某些数据进行打印预览时，由于所预览的图形或数据会超出一屏的显示范围，所以在设计打印预览界面时就需要为预览窗体添加滚动条。通过对滚动条的操作，用户就可以看到超出屏幕之外的数据，程序运行结果如图 10.17 所示。

#### 技术要点

具有滚动条的预览界面主要使用了绘图缓存技术、绘图坐标的定位与滚动条的应用。

- 绘图缓存技术。当在窗体中进行绘图操作时，如果绘图操作过于频繁而且绘制的区域又非常大时，就会出现图片闪耀的现象。这时就需要使用绘图缓存技术使图片的绘制过程不发生闪耀。绘图缓存技术的核心就是将所需要绘制的所有图片或数据绘制在一个图片类中，当绘制完成时再将图片类中所绘制的图片或数据信息一次性的复制到当前画布类中。这样就避免了在画布上绘图时所发生的闪耀现象。

绘图操作都是在窗体或控件的 OnPaint 事件中实现的，所以在该事件的第一行就是获取一个 CPaintDC 类型的画布对象。代码如下：

```
CPaintDC dc(this);
```

这句代码一般情况下是不需要程序员来添加的，而是当映射了重绘消息时由系统自动生成的。接下来的主要任务就是创建一个用于缓存绘图的 CBitmap 图片设备和与图片设备关联的 CDC 画布类。实现代码如下：

```
CPaintDC dc(this);
```

```
CRect rect;
```

```
GetClientRect(&rect);
```

```
//定义图片矩形
```

```
//获取绘图窗口大小
```

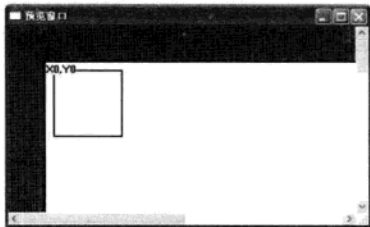


图 10.17 具有滚动条的预览界面

```
CBitmap bitmap; //定义缓存图片
bitmap.CreateCompatibleBitmap(&dc,rect.right,rect.bottom); //创建与当前窗体兼容的图片
CDC memdc; //定义临时画布
memdc.CreateCompatibleDC(NULL); //创建画布
memdc.SelectObject(&bitmap); //关联图片类对象
```

接下来就可以在 memdc 所指向的画布上进行绘制操作了，当所有绘制操作完成后再将 memdc 画布中的内容复制到 dc 画布中。实现代码如下：

```
dc.BitBlt(0,0,rect.right,rect.bottom
&memdc,0,0,SRCCOPY); //画布复制
```

- 绘图坐标的定位。一般情况下绘图都是以窗体可绘图区域的左上角为绘图操作的起点坐标。但这种情况只适用于不带滚动条的绘图窗体，当存在滚动条时绘图的起点坐标应随滚动条的改变而改变。

在该实例中预览窗口中的可绘图区域并不是屏幕的大小，而是打印机默认的纸张大小。为了能更清楚的显示出可绘图区域，在绘图前首先绘制了画布的背景和可绘图区域，如图 10.18 所示。

在图中 x、y 后面的值为绘图的起点坐标。通过这个坐标值可以明显的看出坐标的起点并不是窗体可绘图区域的左上角，而是自定义可绘图区域的左上角。而且细心的读者也会发现，系统坐标系居然不是以窗口的左上角为基准，而是以自定义可绘图区域的左上角为基准。在 Visual C++中要想改变画布的坐标原点可以使用画布类中的 SetViewportOrg 方法，调用该方法将会改变逻辑窗口的原点坐标值。实现代码如下：

```
if (OnDrawPreview != NULL) //存在绘图方法
{
 CRect DrawClientRect(posx,posy,nW-posx,nH-posy); //定义可绘图区域
 memdc.SetViewportOrg(50-posx,50-posy); //设置坐标原点
 OnDrawPreview((CDC*)&memdc,DrawClientRect,isPrint); //调用绘图方法
 memdc.SetViewportOrg(0,0); //将坐标原点设回默认值
 dc.BitBlt(0,0,rect.right,rect.bottom
 ,&memdc,0,0,SRCCOPY); //复制缓存中的绘图信息
}
```

- 滚动条的应用。

给窗体添加滚动条并不是一件复杂的事，首先通过设置窗体的属性让滚动条显示在窗体中。如图 10.19 所示。

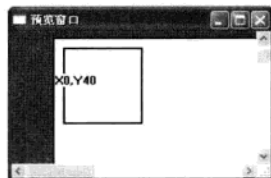


图 10.18 背景与可绘图区域

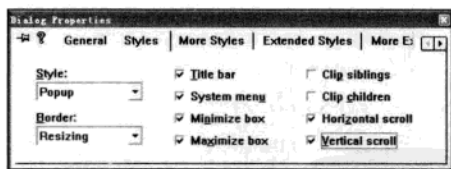


图 10.19 让窗体显示滚动条

还可以通过代码修改窗体的样式来显示滚动条，代码如下：

```
this->ModifyStyle(0,WS_HSCROLL | WS_VSCROLL,0);
```

由于窗体大小的改变会影响滚动条的信息，所以应该在窗体改变消息映射函数中初始化滚动条的信息。滚动条的水平信息或是垂直信息都是通过结构 SCROLLINFO 来完成的，实现代码如下：

```
void CPreviewDialog::SetScrollbar(int cx, int cy)
{
 int vW,vH; //滚动条的水平垂直宽度
 vW = nW + 100; //加上左右边距
 vH = nH + 100; //加上上下边距

 posx = posy = 0; //坐标原点为0

 CRect rect; //窗体客户区
 GetClientRect(&rect);

 SCROLLINFO si; //滚动条结构信息
```



```

si.cbSize = sizeof(si); //结构大小
si.fMask = SIF_ALL; //允许所有操作
si.nMin = 0; //最小值
si.nMax = vH; //内容的高度
si.nPage = rect.Height(); //页面的高度
si.nPos = 0; //当前点
SetScrollInfo(SB_VERT, &si, TRUE); //设置垂直滚动条

si.nMin = 0; //最小值
si.nMax = vW; //内容的宽度
si.nPage = rect.Width(); //页面的宽度
si.nPos = 0; //当前点
SetScrollInfo(SB_HORZ, &si, TRUE); //设置水平滚动条
 }

```

滚动条的初始设置完成后就应添加滚动条的水平和垂直滚动时的消息映射函数了。图 10.20 为添加水平或垂直滚动条的消息映射函数的方法。

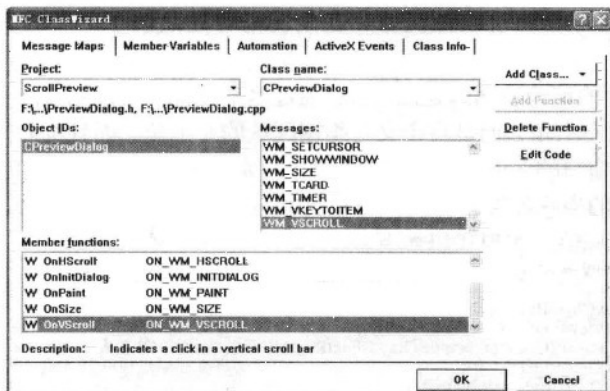


图 10.20 添加滚动条的消息映射方法

滚动条的映射函数主要是获取当前鼠标或键盘对滚动条的操作，然后通过程序对获取的滚动条信息加以修改。实现代码如下：

```

void CPreviewDialog::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
 //垂直滚动条
 switch(nSBCode)
 {
 case SB_TOP: //顶端
 break;
 case SB_BOTTOM: //低端
 break;
 case SB_LINEUP: //按下
 break;
 case SB_LINEDOWN: //抬起
 break;
 case SB_PAGEUP: //上一页
 break;
 case SB_PAGEDOWN: //下一页
 posy = nPos;
 this->Invalidate(false);
 break;
 case SB_THUMBPOSITION: //改变点
 this->SetScrollPos(SB_VERT, nPos);
 break;
 case SB_THUMBTRACK: //改变点
 posy = nPos;
 this->Invalidate(false);
 break;
 }
 CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}

void CPreviewDialog::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
 //水平滚动
 switch(nSBCode)
 {

```

```

 case SB_THUMBTRACK:
 this->SetScrollPos(SB_HORZ,nPos);
 posx = nPos;
 this->Invalidate(false);
 break;
 }
 CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}

```

通过上面的操作就可以实现画布的上下左右移动，但这些操作只能通过鼠标的单击才能完成，若想通过鼠标滑轮实现画布的上下移动就需要映射 OnMouseWheel 消息映射函数来完成，实现代码如下：

```

BOOL CPreviewDialog::OnMouseWheel(UINT nFlags, short zDelta, CPoint pt)
{
 int pos = this->GetScrollPos(SB_VERT);
 if (zDelta < 0)
 pos += 10;
 else
 pos -= 10;
 posy = pos;
 if (posy < 0)
 posy = 0;
 if (posy > nH + 100)
 posy = nH + 100;
 this->SetScrollPos(SB_VERT,pos);
 this->Invalidate(false);
 return CDialog::OnMouseWheel(nFlags, zDelta, pt);
}

```

## 实现过程

(1) 创建一个名为 ScrollPreview 的 MFC 应用程序工程。

(2) 在资源视图中插入一个对话框资源，创建关联类为 CPreviewDialog，ID 为 IDD\_PreviewDialog，该窗体作为预览窗体。

(3) 首先在 CPreviewDialog 类的头文件中定义一个方法指针，该方法指针的作用是指向绘制预览窗体的方法，这样写的好处是通过这个函数指针 CPreviewDialog 类为其他类提供了一个标准的函数接口，通过这个接口任何一个类都可以实现对预览窗体的绘制操作。

定义代码如下：

```

//定义方法指针
typedef void (*DrawPreview)(CDC *PreviewDC, CRect DrawRect, BOOL isPrint = false);

```

(4) CPreviewDialog 在头文件中的部分定义如下：

```

protected:
 int nH,nW;
 int posx,posy;
 BOOL isPrint;
 CDC PrintDC;
 DrawPreview OnDrawPreview;

 void SetScrollbar(int cx, int cy);

public:
 CPreviewDialog(CWnd* pParent = NULL);
 void Show();
 void SetPreviewEvent(DrawPreview OnEvent);

```

(5) 在 CPreviewDialog 类中 SetPreviewEvent 方法用来将其他类实现的打印方法通过参数传递到打印预览类中。实现代码如下：

```

void CPreviewDialog::SetPreviewEvent(DrawPreview OnEvent)
{
 if (OnEvent != NULL)
 OnDrawPreview = OnEvent;
}

```

(6) 当预览窗体初始化时，通过获取打印设备的信息对滚动条进行初始设置，实现代码如下：

```

BOOL CPreviewDialog::OnInitDialog()
{
 CDialog::OnInitDialog();
 CPrintDialog pdlg(false);
 pdlg.GetDefaults();
}

```

```
PrintDC.Attach(pdlg.GetPrinterDC()); //关联打印机画布
CClientDC screencdc(this); //定义屏幕画布
nW = (int)(screencdc.GetDeviceCaps(LOGPIXELSX) //打印机默认纸的宽度
/ 25.4 * PrintDC.GetDeviceCaps(HORZSIZE));
nH = (int)(screencdc.GetDeviceCaps(LOGPIXELSY) //打印机默认纸的高度
/ 25.4 * PrintDC.GetDeviceCaps(VERTSIZE));

this->ModifyStyle(0, WS_HSCROLL | WS_VSCROLL, 0); //添加水平垂直滚动条

int vW, vH;
vW = nW + 100;
vH = nH + 100;

this->SetScrollRange(SB_HORZ, 0, vW, true); //设置水平滚动条区域
this->SetScrollRange(SB_VERT, 0, vH, true); //设置垂直滚动条区域

posx = 0; //原点坐标
posy = 0;

return TRUE;
}
```

(7) 当窗体大小发生改变时, 对滚动条的信息进行修改。实现代码如下:

```
void CPreviewDialog::OnSize(UINT nType, int cx, int cy)
{
 CDialog::OnSize(nType, cx, cy);
 SetScrollbar(cx, cy); //设置滚动条信息
 this->Invalidate(false); //重画
}
```

(8) 当窗体发生重画事件时绘制窗体背景和可绘图区域, 并调用打印预览的绘制方法进行绘图操作。实现代码如下:

```
void CPreviewDialog::OnPaint()
{
 CPaintDC dc(this);
 CRect rect;
 GetClientRect(&rect); //获取客户区域
 CBitmap bitmap; //定义图片类
 bitmap.CreateCompatibleBitmap(&dc, rect.right, rect.bottom); //创建关联图片
 CDC memdc; //定义临时画布
 memdc.CreateCompatibleDC(NULL); //创建画布
 memdc.SelectObject(&bitmap); //关联图片
 CBrush *b; //定义画刷
 b = (memdc.GetHalftoneBrush()); //获取背景画刷
 memdc.SelectObject(b); //选择画刷
 rect.InflateRect(5, 5, 5, 5); //扩大矩形
 memdc.Rectangle(&rect); //绘制矩形

 CPen pen(BS_SOLID, 2, RGB(0, 0, 0)); //定义黑色画笔
 CBrush brush(RGB(255, 255, 255)); //定义白色画刷
 memdc.SelectObject(&pen); //选择画笔
 memdc.SelectObject(&brush); //选择画刷
 CRect DrawRect(50-posx, 50-posy, 50-posx+nW, 50-posy+nH); //定义可绘图区域
 DrawRect.InflateRect(2, 2, 2, 2); //扩大矩形
 memdc.Rectangle(&DrawRect); //绘制可绘图区域

 if (OnDrawPreview != NULL) //存在绘图方法
 {
 CRect DrawClientRect(posx, posy, nW-posx, nH-posy); //定义绘图区域
 memdc.SetViewportOrg(50-posx, 50-posy); //设置坐标原点
 OnDrawPreview((CDC*)&memdc, DrawClientRect, isPrint); //调用绘图方法
 memdc.SetViewportOrg(0, 0); //恢复坐标原点
 dc.BitBlt(0, 0, rect.right, rect.bottom, &memdc, 0, 0, SRCCOPY); //复制图片
 }
}
```

## 举一反三

根据本实例, 读者可以:

- 模拟 Word 2000 的打印预览。

## 实例 320 在对话框中分页预览

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\10\320

### 实例说明

在 VisualC++ 环境中多文档界面的应用程序中给出了打印及打印预览的类，程序设计人员可以调用打印或打印预览的类进行数据或图形的打印及预览操作。而要实现对话框的打印及预览功能就应由程序员自己来实现对数据及图形的打印及预览操作，并且应具备分页预览的功能。如图 10.21 所示。

### 技术要点

具有滚动条的预览界面主要使用了绘图缩放技术、绘图缓存技术、分页预览的实现与鼠标滑轮控制垂直滚动条及缩放。

- 绘图缩放技术。绘图缩放技术是绘图过程中的一项非常重要的技术，有许多的程序员在对绘图缩放操作时还在想怎样对坐标位置、线的宽度及文字的大小进行设置

来达到缩放效果，或者将绘制好的图形缩放复制到比较大或者比较小的画布上。这些做法都及其影响了绘图操作的质量。最好的办法应该是使用 `SetWindowExt` 方法和 `SetViewportExt` 方法来改变物理窗体与逻辑窗口的大小，从而实现窗口缩放的效果。并且在使用这两个方法前应先使用 `SetMapMode` 方法来设置单位的映射模式。单位映射模式的类型由表 10.2 所示。



图 10.21 在对话框中分页预览

表 10.2 单位映射模式的分类及说明

| 单位映射模式的分类      | 说 明                                                                                                                                                    |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| MM_ANISOTROPIC | 把逻辑单位转换为任意缩放轴上的任意单位。把映射模式设置为 <code>MM_ANISOTROPIC</code> 并不改变当前窗口或视图端口的设置。要改变单位、方向和缩放，可调用 <code>SetViewportExt</code> 和 <code>SetWindowExt</code> 成员函数 |
| MM_HIENGLISH   | 每一逻辑单位对换 0.001 英寸，x 向右为正，y 向上为正                                                                                                                        |
| MM_HIMETRIC    | 每一逻辑单位对换 0.001 毫米，x 向右为正，y 向上为正                                                                                                                        |
| MM_ISOTROPIC   | 逻辑单位转换为带有对等缩放轴的任意单位。即 x 轴 1 单位与 y 轴 1 单位是相等的。可使用 <code>SetViewportExt</code> 和 <code>SetWindowExt</code> 成员函数，指定需要的单位和轴的方向。GDI 修正可以保证 x、y 轴的尺寸是一致的     |
| MM_LOENGLISH   | 每一逻辑单位对换 0.01 英寸，x 向右为正，y 向上为正                                                                                                                         |
| MM_LOMETRIC    | 每一逻辑单位对换 0.1 毫米，x 向右为正，y 向上为正                                                                                                                          |
| MM_TEXT        | 每一逻辑单位对换 1 设备像素，x 向右为正，y 向下为正                                                                                                                          |
| MM_TWIPS       | 每一逻辑单位对换 1/20 个点 (1 点为 1/72 英寸，1twip 为 1/1440 英寸)，x 向右为正，y 向上为正                                                                                        |

在该实现中预览窗口的缩放应保持水平与垂直的等比例缩放，所以应使用 `MM_ISOTROPIC` 映射模式。实现代码如下：

```
dc.SetMapMode(MM_ISOTROPIC); //设置映射模式
dc.SetWindowExt(100,100); //物理窗体大小
dc.SetViewportExt(Zoom,Zoom); //逻辑窗体大小
```



当变量 Zoom 的值大于 100 时绘制出来的图形是放大的, 当变量 Zoom 的值小于 100 时绘制出来是图形是缩小的。

- 绘图缓存技术。当在窗体中进行绘图操作时, 如果绘图操作过于频繁而且绘制的区域又非常大时, 就会出现图片闪耀的现象。这就需要使用绘图缓存技术使图片的绘制过程不发生闪耀。绘图缓存技术的核心就是将所需要绘制的所有图片或数据绘制在一个图片类中, 当绘制完成时再将图片类中所绘制的图片或数据信息一次性的复制到当前画布类中。这样就避免了在画布上绘图时所发生的闪耀现象。

绘图操作都是在窗体或控件的 OnPaint 事件中实现的, 所以在该事件的第一行就是获取一个 CPaintDC 类型的画布对象。代码如下:

```
CPaintDC dc(this);
```

这句代码一般情况下是不需要程序员来添加的, 当映射了重绘消息时由系统自动生成的。接下来的主要任务就是创建一个用于缓存绘图的 CBitmap 图片设备和与图片设备关联的 CDC 画布类。实现代码如下:

```
CRect ClientRect; //定义客户矩形
this->GetParent()->GetClientRect(&ClientRect); //获取客户区
ClientRect.InflateRect(2,2,2,2); //扩大
if (Zoom < 100)
 dc.DPtoLP(&ClientRect); //转换单位
CRect rect(FrameMargin.left,FrameMargin.top,
 PageSize.cx+FrameMargin.left,PageSize.cy+FrameMargin.top);
CBitmap bitmap; //定义图片类
bitmap.CreateCompatibleBitmap(&dc,ClientRect.right,ClientRect.bottom); //创建图片
CDC MemDC; //定义临时画布
MemDC.CreateCompatibleDC(NULL); //创建画布
MemDC.SelectObject(&bitmap); //关联图像
```

接下来就可以在 MemDC 所指向的画布上进行绘制操作了, 当所有绘制操作完成后将 MemDC 画布中的内容复制到 dc 画布中。实现代码如下:

```
dc.BitBlt(0,0,ClientRect.right,ClientRect.bottom,&MemDC,0,0,SRCCOPY); //复制图像
```

- 分页预览的实现。分页预览是打印预览所应具备的基本功能, 分页将打印的数据或图形分成若干个页面进行显示。最简单的分页就是本实例所实现的单页切换预览, 这种方式的分页显示方式使每页显示在独立的预览窗口中, 并且不能在同一预览窗口中显示两页的内容。所以这种预览方式在绘图方面比较简单, 更容易实现。在本实例中分页操作可通过按钮和鼠标滑轮来实现。

在该实例中变量 PageCount 记录了所预览的总页数, 变量 CurrentPage 记录了当前预览的页数, 预览页就是通过对 CurrentPage 变量的加 1 或减 1 操作来实现上一页或下一页的切换的。同时绘图操作则根据 CurrentPage 变量的值绘制不同的预览页面。实现代码如下:

```
//上一页
void CPreviewDialog::OnPrev()
{
 if (CurrentPage > 1)
 {
 CurrentPage -= 1; //页减1
 this->Invalidate(false); //重画
 UpdateStatusBar(); //更新状态栏
 }
}

//下一页
void CPreviewDialog::OnNext()
{
 if (CurrentPage < PageCount)
 {
 CurrentPage += 1; //页加1
 this->Invalidate(false); //重画
 UpdateStatusBar(); //更新状态栏
 }
}
```

通过鼠标滑轮来控制页面的切换则是判断当前滚动条滑块的位置是否小于 0 或大于滑块的最大位置来实现。当小于 0 时切换到上一页, 否则切换到下一页。实现代码如下:

```
int pos = this->GetScrollPos(SB_VERT); //获取滚动条位置
if (zDelta < 0)
 pos += 10;
else
 pos -= 10;

if (pos < 0)
{
 OnPrev(); //上一页
 pos = vp.cx * (Zoom / 100) - vp.cy; //定义页面末端
}
else
if (pos > vp.cx * (Zoom / 100) - vp.cy)
{
 OnNext(); //下一页
 pos = 0; //定位页面顶端
}
}
```

- 鼠标滑轮控制垂直滚动条及缩放。提供鼠标滑轮的操作是为了方便用户操作预览界面的。用户只需通过鼠标滑轮就可以实现页面的上移、下移、翻页、缩放操作。需要说明的是缩放操作是配合键盘上的<Ctrl>键来完成的。实现代码如下：

```
BOOL CPreviewDialog::OnMouseWheel(UINT nFlags, short zDelta, CPoint pt)
{
 if (nFlags & MK_CONTROL) //按下Ctrl键
 {
 if (zDelta < 0)
 OnZoomOut(); //放大
 else
 OnZoomIn(); //缩小
 }
 else
 {
 int pos = this->GetScrollPos(SB_VERT); //获取滚动条位置
 if (zDelta < 0)
 pos += 10;
 else
 pos -= 10;

 if (pos < 0)
 {
 OnPrev(); //上一页
 pos = vp.cx * (Zoom / 100) - vp.cy;
 }
 else
 if (pos > vp.cx * (Zoom / 100) - vp.cy)
 {
 OnNext(); //下一页
 pos = 0;
 }
 StartPoint.y = pos;
 this->SetScrollPos(SB_VERT, pos); //修改滚动条新位置
 this->Invalidate(false); //重画
 }
 return CDialog::OnMouseWheel(nFlags, zDelta, pt);
}
```

## 实现过程

- (1) 创建一个名为 PagesPreview 的 MFC 应用程序工程。
- (2) 在资源视图中插入一个对话框资源，创建关联类为 CPreView，ID 为 IDD\_Preview，该窗体作为预览窗体的框架窗体。在该窗体中将实现工具栏、状态栏等功能。
- (3) 在资源视图中插入一个对话框资源，创建关联类为 CPreviewDialog，ID 为 IDD\_PreviewView，该窗体作为预览窗体。该窗体将作为子窗体嵌入在框架窗体中。
- (4) 当 CPreviewDialog 类所关联的对话框进行初始化时初始化预览窗口所需要的信息。实现代码如下：

```
BOOL CPreviewDialog::OnInitDialog()
{
 CDialog::OnInitDialog();

 PageCount = 3; //总页数
 CurrentPage = 1; //当前页
}
```

```
Zoom = 100; //缩放
CClientDC dc(this);
Margin = CRect(1500,1500,1500,1500); //页边距
printsetup = new CPrintDialog(false); //打印对话框
printsetup->GetDefaults(); //获取打印机默认设置
InitPreview(printsetup->GetPrinterDC(),*printsetup->GetDevMode()); //初始化页面信息
pagesetup = new CPageSetupDialog(); //页面设置对话框

return TRUE;
```

(5) InitPreview 方法用来初始化预览页的大小及页边距的大小, 实现代码如下:

```
void CPreviewDialog::InitPreview(HDC hDC,DEVMODE dev)
{
 CDC pdc;
 CClientDC sdc(this); //屏幕画布
 pdc.Attach(hDC); //关联打印机
 PageSize.cx = (int)(sdc.GetDeviceCaps(LOGPIXELSX) /
 25.4 * pdc.GetDeviceCaps(HORZSIZE)); //打印页宽度
 PageSize.cy = (int)(sdc.GetDeviceCaps(LOGPIXELSY) /
 25.4 * pdc.GetDeviceCaps(VERTSIZE)); //打印页高度
 FrameMargin = CRect(20,20,20,20); //页边距
 pdc.Detach(); //取消关联
 pdc.DeleteDC(); //删除DC
 UpdateScroll(); //更新滚动条
}
```

(6) 当页面大小计算完成后就会调用 UpdateScroll 方法来设置滚动条的相关信息, 实现代码如下:

```
void CPreviewDialog::UpdateScroll()
{
 CRect rect;
 GetClientRect(&rect); //窗体客户区
 CClientDC dc(this); //画布
 dc.SetMapMode(MM_ISOTROPIC); //映射模式
 dc.SetWindowExt(100,100); //物理窗口大小
 dc.SetViewportExt(Zoom,Zoom); //缩放比例

 hp.cx = (PageSize.cx + FrameMargin.left + FrameMargin.right); //水平总宽度
 hp.cy = (rect.right); //页宽度
 dc.DPtoLP(&hp); //单位转换

 vp.cx = (PageSize.cy + FrameMargin.top + FrameMargin.bottom); //垂直总高度
 vp.cy = (rect.bottom); //页高度
 dc.DPtoLP(&vp); //单位转换

 StartPoint.x = StartPoint.y = 0; //原点坐标
 SCROLLINFO scinfo;
 scinfo.cbSize = sizeof(SCROLLINFO);
 scinfo.fMask = SIF_ALL;
 scinfo.nMin = 0;
 scinfo.nMax = hp.cx * (Zoom / 100); //水平总宽度
 scinfo.nPage = hp.cy; //页宽度
 scinfo.nPos = 0;

 this->SetScrollInfo(SB_HORZ,&scinfo); //设置水平滚动条

 scinfo.nMax = vp.cx * (Zoom / 100); //垂直总高度
 scinfo.nPage = vp.cy; //页高度
 this->SetScrollInfo(SB_VERT,&scinfo); //设置垂直滚动条
 this->Invalidate(false); //重画
}
```

(7) 当预览窗口的 OnPaint 方法被调用时将跟据当前页数的不同绘制不同的预览页, 实现代码如下:

```
void CPreviewDialog::OnPaint()
{
 CPaintDC dc(this); //定义画布

 dc.SetMapMode(MM_ISOTROPIC); //设置映射模式
 dc.SetWindowExt(100,100); //物理窗口大小
 dc.SetViewportExt(Zoom,Zoom); //缩放比例

 CRect ClientRect;
 this->GetParent()->GetClientRect(&ClientRect); //获取客户区
 ClientRect.InflateRect(2,2,2,2); //扩大
 if (Zoom < 100)
 dc.DPtoLP(&ClientRect); //单位转换
 CRect rect(FrameMargin.left,FrameMargin.top,
```

```

 PageSize.cx+FrameMargin.left,PageSize.cy+FrameMargin.top); //可绘图区域

 CBitmap bitmap; //图片类
 bitmap.CreateCompatibleBitmap(&dc,ClientRect.right,ClientRect.bottom); //创建兼容图片
 CDC MemDC; //临时画布
 MemDC.CreateCompatibleDC(NULL); //创建画布
 MemDC.SelectObject(&bitmap); //关联图片

 CBrush BkBrush(RGB(157,155,151)); //定义背景画刷
 MemDC.SelectObject(&BkBrush); //选择画刷
 MemDC.Rectangle(&ClientRect); //绘制背景

 CBrush brush(RGB(255,255,255)); //定义白色画刷
 CPen pen(BS_SOLID,2,RGB(0,0,255)); //定义画笔
 MemDC.SelectObject(&brush); //选择画刷
 MemDC.SelectObject(&pen); //选择画笔
 rect.OffsetRect(-StartPoint.x,-StartPoint.y); //移动可绘图区
 rect.InflateRect(1,1,1,1); //扩大矩形
 MemDC.Rectangle(&rect); //绘制矩形
 rect.InflateRect(-1,-1,-1,-1); //缩小矩形

 CPen p(BS_SOLID,1,RGB(0,0,0)); //定义黑色画笔
 MemDC.SelectObject(&p); //选择画笔

 double ratex = dc.GetDeviceCaps(LOGPIXELSX) / 25.4; //与打印机水平方向缩放比例
 double ratey = dc.GetDeviceCaps(LOGPIXELSY) / 25.4; //与打印机垂直方向缩放比例
 CRect Margins = CRect((int)(ratex * (Margin.left / 100)), //页边距
 (int)(ratey * (Margin.top / 100)),
 (int)(ratex * (Margin.right / 100)),
 (int)(ratey * (Margin.bottom / 100)));

 CRect DrawRect(0,0,PageSize.cx - Margins.left - Margins.right,
 PageSize.cy - Margins.top - Margins.bottom); //绘图区域
 DrawRect.OffsetRect(-StartPoint.x,-StartPoint.y); //移动绘图区域到原点

 MemDC.SetViewportOrg(FrameMargin.left + Margins.left,
 FrameMargin.top + Margins.top); //设置逻辑窗口原点坐标

 MemDC.IntersectClipRect(&DrawRect); //将绘图区域设为剪裁区

 DrawPreview(&MemDC,&DrawRect); //绘制页
 MemDC.SetViewportOrg(0,0); //恢复逻辑窗口坐标原点

 dc.BitBlt(0,0,ClientRect.right,ClientRect.bottom,&MemDC,0,0,SRCCOPY);
 MemDC.DeleteDC(); //复制临时画布到预览窗口
 bitmap.DeleteObject(); //删除图片
}

```

(8) DrawPreview 是一个虚方法，该方法用来实现对预览页中可绘图区域的绘制，在使用时可通过子类实现其绘制方法。实现代码如下：

```

void CPreviewDialog::DrawPreview(CDC *DrawDC,CRect *DrawRect)
{
 if (CurrentPage == 1) //第1页
 DrawDC->Rectangle(DrawRect); //绘图
 else if (CurrentPage == 2) //第2页
 DrawDC->Ellipse(DrawRect); //绘图
 else //第3页
 {
 DrawDC->DrawEdge(DrawRect,EDGE_BUMP,BF_RECT); //绘图
 }
}

```

(9) 在该实例中不但实现了滚动条的拖动操作，还实现了行操作、翻页操作等，实现代码如下：

```

//水平滚动条
void CPreviewDialog::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
 CRect rect;
 this->GetClientRect(&rect); //客户区
 switch(nSBCode)
 {
 case SB_LINELEFT: //向左移动一行
 StartPoint.x -= 10;
 if (StartPoint.x < 0)
 StartPoint.x = 0;
 this->SetScrollPos(SB_HORZ,StartPoint.x); //修改滚动条位置
 this->Invalidate(false); //重画
 break;
 case SB_LINERIGHT: //向右移动一行
 StartPoint.x += 10;

```



```

 if (StartPoint.x > hp.cx * (Zoom / 100) - hp.cy)
 StartPoint.x = hp.cx * (Zoom / 100) - hp.cy;
 this->SetScrollPos(SB_HORZ, StartPoint.x); //修改滚动条位置
 this->Invalidate(false); //重画
 break;
 case SB_PAGELEFT: //向左翻页
 StartPoint.x -= rect.right;
 if (StartPoint.x < 0)
 StartPoint.x = 0;
 this->SetScrollPos(SB_HORZ, StartPoint.x);
 this->Invalidate(false);
 break;
 case SB_PAGEDOWN: //向右翻页
 StartPoint.x += rect.right;
 if (StartPoint.x > hp.cx * (Zoom / 100) - hp.cy)
 StartPoint.x = hp.cx * (Zoom / 100) - hp.cy;
 this->SetScrollPos(SB_HORZ, StartPoint.x);
 this->Invalidate(false);
 break;
 case SB_THUMBTRACK: //滚动条拖动
 StartPoint.x = nPos;
 this->SetScrollPos(SB_HORZ, nPos);
 this->Invalidate(false);
 break;
 }
 CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
//垂直滚动条
void CPreviewDialog::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
 CRect rect;
 this->GetClientRect(&rect); //获取客户区
 switch(nSBCode)
 {
 case SB_LINEUP: //向上一行
 StartPoint.y -= 10;
 if (StartPoint.y < 0)
 StartPoint.y = 0;
 this->SetScrollPos(SB_VERT, StartPoint.y);
 this->Invalidate(false);
 break;
 case SB_LINEDOWN: //向下一行
 StartPoint.y += 10;
 if (StartPoint.y > vp.cx * (Zoom / 100) - vp.cy)
 StartPoint.y = vp.cx * (Zoom / 100) - vp.cy;
 this->SetScrollPos(SB_VERT, StartPoint.y);
 this->Invalidate(false);
 break;
 case SB_PAGEUP: //上翻页
 StartPoint.y -= rect.bottom;
 if (StartPoint.y < 0)
 StartPoint.y = 0;
 this->SetScrollPos(SB_VERT, StartPoint.y);
 this->Invalidate(false);
 break;
 case SB_PAGEDOWN: //下翻页
 StartPoint.y += rect.bottom;
 if (StartPoint.y > vp.cx * (Zoom / 100) - vp.cy)
 StartPoint.y = vp.cx * (Zoom / 100) - vp.cy;
 this->SetScrollPos(SB_VERT, StartPoint.y);
 this->Invalidate(false);
 break;
 case SB_THUMBTRACK: //拖动滚动条
 StartPoint.y = nPos;
 this->SetScrollPos(SB_VERT, nPos);
 this->Invalidate(false);
 break;
 }
 CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}

```

根据本实例，读者可以：

- 自定义按比例显示的打印预览窗口。

# 第 11 章

## 硬件相关开发技术

- 串口控制
- 加密狗和加密锁
- IC 卡、ID 卡应用
- 监控
- 扫描、条形码、POS 控制
- 语音卡控制
- 手机程序开发
- 其他程序

Visual C++

资源分享

PDG

## 11.1 串口控制

串行口是计算机的标准接口, 现在的 PC 机器一般至少有两个串行口 COM1 和 COM2。串行口应用非常广泛, 在数据通信、计算机网络及分布式工业控制系统中, 经常采用串行通信来交换数据和信息。本节通过几个实例来介绍串口的应用技术和方法。

### 实例 321 通过串口传递数据

这是一个可以提高基础性能的实例

实例位置: 光盘\mingrisoft\11\321

#### 实例说明

如今的许多硬件设备都采用串口技术与计算机相连, 因此, 串口应用程序开发越来越普遍。本例实现了两台计算机间的串口通信。实例运行结果如图 11.1 所示。

#### 技术要点

在 Win32 环境下, 串口资源是作为文件来对待的。因此可以将串口和其他通讯设备作为文件来处理。例如, 打开一个串口, 可以使用 CreateFile 函数; 发送一个串口数据, 可以使用 WriteFile 函数; 读取串口数据, 可以使用 ReadFile 函数。下面介绍与串口有关的 API 函数。

(1) CreateFile 函数。该函数用于打开一个串口。语法如下:

```
HANDLE CreateFile(LPCTSTR lpFileName, DWORD dwDesiredAccess,
DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes, DWORD dwCreationDisposition, DWORD
dwFlagsAndAttributes, HANDLE hTemplateFile);
```

参数说明:

- lpFileName: 标识打开的逻辑端口名, 例如“COM1”、“COM2”等。
- DwDesiredAccess: 确定串口的访问模式, 可选值如下。
  - GENERIC\_READ: 允许对设备进行读访问。
  - GENERIC\_WRITE: 允许对设备进行写访问。
- dwShareMode: 标识共享模式, 可选值如下。
  - 0: 表示不共享。对于串口, dwShareMode 参数只能设置为 0。
  - FILE\_SHARE\_READ: 对文件进行读共享。
  - FILE\_SHARE\_WRITE: 对文件进行写共享。
  - FILE\_SHARE\_DELETE: 对文件进行删除共享。
- lpSecurityAttributes: 一个 SECURITY\_ATTRIBUTES 结构指针, 标识文件的安全属性, 对于串口, 该参数应设置为 NULL。
- dwCreationDisposition: 标识文件创建方式, 可选值如下。
  - CREATE\_NEW: 创建一个新文件, 如果文件已经存在, 将出现错误。
  - CREATE\_ALWAYS: 创建一个新文件, 如果文件已经存在, 覆盖原来的文件。
  - OPEN\_EXISTING: 打开一个存在的文件。如果文件不存在, 将会出现错误。
  - OPEN\_ALWAYS: 打开一个文件, 如果文件不存在, 将创建一个文件。
  - TRUNCATE\_EXISTING: 打开一个文件, 将文件长度设置为 0。如果文件不存在, 函数将执行失败。



图 11.1 通过串口传递数据

- **DwFlagsAndAttributes**: 该参数用于设置文件标记和属性。可选值如下。
  - **FILE\_ATTRIBUTE\_ARCHIVE**: 标记归档属性。
  - **FILE\_ATTRIBUTE\_HIDDEN**: 标识文件是隐藏的。
  - **FILE\_ATTRIBUTE\_NORMAL**: 标识文件的正常属性。该标记不能和其他标记组合使用。
  - **FILE\_ATTRIBUTE\_OFFLINE**: 标识文件数据不能立即使用。
  - **FILE\_ATTRIBUTE\_READONLY**: 标识文件是只读的。
  - **FILE\_ATTRIBUTE\_SYSTEM**: 标识文件是系统文件。
  - **FILE\_ATTRIBUTE\_TEMPORARY**: 标识文件是临时文件。
  - **FILE\_FLAG\_WRITE\_THROUGH**: 标识系统不通过缓存, 直接写文件到磁盘中。
  - **FILE\_FLAG\_OVERLAPPED**: 允许对文件进行异步操作。
  - **FILE\_FLAG\_NO\_BUFFERING**: 对文件进行缓冲管理。
  - **FILE\_FLAG\_RANDOM\_ACCESS**: 对随机访问的文件进行缓冲优化。
  - **FILE\_FLAG\_SEQUENTIAL\_SCAN**: 对连续访问的文件进行缓冲优化。
  - **FILE\_FLAG\_DELETE\_ON\_CLOSE**: 当文件句柄关闭后, 删除文件。一般适用于临时文件。
  - **SECURITY\_ANONYMOUS**: 以匿名的方式模拟客户端。
  - **SECURITY\_IDENTIFICATION**: 以署名的方式模拟客户端。
  - **SECURITY\_IMPERSONATION**: 以扮演者身份模拟客户端。
  - **SECURITY\_DELEGATION**: 以代理方式模拟客户端。
  - **SECURITY\_CONTEXT\_TRACKING**: 标识安全模式是动态的。
  - **SECURITY\_EFFECTIVE\_ONLY**: 客户端的安全描述部分对服务器可用。
- **hTemplateFile**: 标识一个文件句柄, 如果不为零, 新文件将从该句柄标识的文件中复制扩展属性。如果函数执行成功, 返回值是打开的串口句柄。

(2) **SetupComm** 函数。该函数用于设置输入缓冲区和输出缓冲区的大小。语法如下:

```
BOOL SetupComm(HANDLE hFile, DWORD dwInQueue, DWORD dwOutQueue);
```

参数说明:

- **hFile**: 标识串口句柄, 通常为 **CreateFile** 的返回值。
- **dwInQueue**: 标识输入缓冲区大小。
- **dwOutQueue**: 标识输出缓冲区大小。

(3) **SetCommState** 函数。该函数用于设置串口状态。语法如下:

```
BOOL SetCommState(HANDLE hFile, LPDCB lpDCB);
```

参数说明:

- **hFile**: 表示串口句柄。
- **lpDCB**: DCB 结构指针, 函数通过该参数设置串口信息。用户在使用该函数前, 可以调用 **GetCommState** 函数获取当前串口的配置信息; 然后对 DCB 结构的成员进行设置, 这样, 对于不需要设置的串口信息, 就不用设置了。

(4) **ReadFile** 函数。该函数用于读取串口数据。语法如下:

```
BOOL ReadFile(HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped);
```

参数说明:

- **hFile**: 标识串口句柄。
- **lpBuffer**: 标识数据接收缓冲区。
- **nNumberOfBytesToRead**: 标识要从串口读取的字节数。
- **lpNumberOfBytesRead**: 标识实际读取的字节数。
- **lpOverlapped**: 一个 **OVERLAPPED** 结构指针, 该结构包含了异步操作时的输入、输出信息。





(5) WriteFile 函数。该函数用于向串口中写入数据。语法如下:

BOOL WriteFile(HANDLE hFile,LPCVOID lpBuffer,DWORD nNumberOfBytesToWrite,  
LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped);

参数说明:

- hFile: 标识串口句柄。
- lpBuffer: 标识数据发送缓冲区。
- nNumberOfBytesToRead: 标识要发送到串口的字节数。
- lpNumberOfBytesRead: 标识实际写入串口的字节数。lpOverlapped 是一个 OVERLAPPED 结构指针, 该结构包含了异步操作时的输入、输出信息。

(6) WaitCommEvent 函数。该函数用于检测通信设备中发生的通信事件。语法如下:

BOOL WaitCommEvent(HANDLE hFile,LPDWORD lpEvtMask, LPOVERLAPPED lpOverlapped,);

参数说明:

- hFile: 标识串口句柄。
- lpEvtMask: 用于接收事件掩码。
- lpOverlapped: OVERLAPPED 结构指针, 该结构中包含了异步操作的相关信息。

(7) CreateEvent 函数。该函数用于创建一个命名的或未命名的事件对象。语法如下:

HANDLE CreateEvent(LPSECURITY\_ATTRIBUTES lpEventAttributes,BOOL bManualReset, BOOL bInitialState,LPCTSTR lpName);

参数说明:

- lpEventAttributes: 一个指向 SECURITY\_ATTRIBUTES 结构的指针, 该结构描述了安全性信息。如果参数为 NULL, 将使用默认的安全性。
- bManualReset: 用于指定创建人工重置事件对象, 还是自动重置对象, 如果为 TRUE, 表示创建人工重置对象。
- bInitialState: 表示创建事件时的初始状态, 如果为 TRUE, 表示有信号状态。
- lpName: 用于指定事件的名称, 如果为 NULL, 将创建一个匿名事件对象。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加按钮控件、静态文本控件、文本编辑控件。
- (3) 处理“打开串口”按钮的单击事件, 打开串口。

void CSerialPortDlg::OnOK()

```
{
 hCom = CreateFile("COM1", //打开串口1
 GENERIC_READ|GENERIC_WRITE, //允许读和写操作
 0, //独占方式
 NULL,
 OPEN_EXISTING, //打开一个存在的串口
 FILE_ATTRIBUTE_NORMAL|FILE_FLAG_OVERLAPPED, //异步方式打开
 NULL
);
 if (hCom==INVALID_HANDLE_VALUE)
 {
 MessageBox("端口打开失败.");
 return;
 }
}
```

- (4) 定义一个线程函数, 读取串口中的数据。

//线程函数

DWORD ThreadFunction(LPVOID pParam)

```
{
 DWORD dwEvtMask,dwResult;
 tOverLaped.hEvent = CreateEvent(NULL,TRUE,FALSE,NULL); //创建一个事件
 while (IsFun)
 {
 //等待窗口事件
 }
}
```

```
WaitCommEvent(((CSerialPortDlg*)AfxGetMainWnd())->hCom,
&dwEvtMask,&tOverLaped);
//如果事件没有信号, 延时0.1秒
dwResult = WaitForSingleObject(tOverLaped.hEvent,100);
switch (dwResult)
{
case WAIT_OBJECT_0: //事件对象有信号
 switch (dwEvtMask)
 {
 case EV_RXCHAR: //接收到数据
 {
 if (IsStop)//发送停止
 {
 IsStop = FALSE;
 //发送消息, 由消息处理函数接收数据
 ::PostMessage(AfxGetMainWnd()->m_hWnd,
CM_RECEIVE,0,(LPARAM)EV_RXCHAR);
 }
 break;
 }
 }
}
}
return 0;
}
```

(5) 处理“设置串口”按钮的单击事件, 设置串口信息, 同时创建一个线程, 检测串口发生的事件。

```
void CSerialPortDlg::OnButton2()
{
 SetupComm(hCom,1024,1024); //设置发送和接收缓冲区大小
 //设置串口信息
 DCB dcb;
 GetCommState(hCom,&dcb);
 dcb.BaudRate = 9600;
 dcb.fBinary = TRUE;
 dcb.fParity = TRUE;
 dcb.ByteSize = 8;
 dcb.Parity = ODDPARITY;

 dcb.StopBits = ONESTOPBIT;

 if (!SetCommState(hCom,&dcb))//设置串口状态
 {
 MessageBox("设置失败.", "提示");
 return;
 }

 if (!SetCommMask(hCom,EV_RXCHAR | EV_TXEMPTY)//设置串口掩码
 {
 MessageBox("掩码设置失败.", "提示");
 return;
 }

 DWORD param;
 hThread = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
ThreadFunction,¶m,0,&dwThreadId);//创建线程
 if (hThread==INVALID_HANDLE_VALUE)
 {
 MessageBox("线程创建失败.", "提示",64);
 return;
 }
 IsFun = TRUE;
}
```

(6) 在程序中自定义一个消息, 并编写消息处理函数。当线程函数发现串口中有数据读取时会向应用程序发送消息, 执行消息处理函数。

```
//自定义一个消息
const CM_RECEIVE = WM_USER+100;
//添加消息映射宏
ON_MESSAGE(CM_RECEIVE,OnRecieveData)
//自定义消息处理函数, 用于接收数据
void CSerialPortDlg::OnRecieveData(WPARAM wParam, LPARAM lParam)
```

```
{
 DWORD res, factbyte;

 memset(DataBuffer, 0, 1024); //初始化数据缓冲区
 COMSTAT rst;

 //清空串口错误标志，记录当前通信状态
 ClearCommError(hCom, &res, &rst);
 //创建一个事件对象
 rOverLaped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
 //读取数据到缓冲区中
 if (ReadFile(hCom, DataBuffer, rst.cbInQue, &factbyte, &rOverLaped))
 {
 DataBuffer[rst.cbInQue] = 0;
 IsStop = FALSE;
 }
 else
 {
 res = WaitForSingleObject(rOverLaped.hEvent, 5000);
 }
 IsStop = FALSE;
 MessageBox(DataBuffer);
}
```

### 举一反三

根据本实例，读者可以：

- 远程监控对方计算机屏幕。

## 实例 322 通过串口控制对方计算机 关闭

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\11\322

### 实例说明

在网络应用程序中，主要通过网卡实现数据的传输，因此可以利用套接字技术实现远程关闭计算机。如果计算机中没有安装网卡，该如何实现远程关闭计算机呢？本例实现了利用串口关闭对方计算机，程序运行结果如图 11.2 所示。

### 技术要点

本例使用了 ActiveX 控件——MSComm 实现串口的通信。首先需要导入 MSComm 控件。用鼠标右键单击对话框，在弹出的快捷菜单中选择 Insert ActiveX Control 菜单项，打开 Insert ActiveX Control 窗口，在该窗口中选择 Microsoft Common Dialog Control 选项，单击“OK”按钮导入 MSComm 控件。通过类向导为 MSComm 控件命名，同时导入 CMSComm 类。CMSComm 类提供了多个方法用于控制串口。其主要方法、事件如下。

(1) SetCommPort 方法。该方法用于设置端口号。语法如下：

```
void SetCommPort(short nNewValue);
```

参数说明：

- nNewValue：标识端口号。

(2) SetRThreshold 方法。该方法用于设置收到多少个字符后触发 OnComm 事件。语法如下：

```
void SetRThreshold(short nNewValue);
```

参数说明：

- nNewValue：标识字符数。

(3) SetPortOpen 方法。该方法用于打开端口。语法如下：



图 11.2 通过串口控制对方计算机关闭

```
void SetPortOpen(BOOL bNewValue);
```

参数说明:

- nNewValue: 标识打开的端口号。

(4) SetOutput 方法。该方法用于向串口发送数据。语法如下:

```
void SetOutput(const VARIANT& newValue);
```

参数说明:

- newValue: 标识发送的数据。

(5) GetInput 方法。该方法用于从串口获取数据。语法如下:

```
VARIANT GetInput();
```

返回值: 从串口接收的数据。

(6) GetCommEvent 方法。该方法用于获取串口中的事件。语法如下:

```
short GetCommEvent();
```

返回值: 返回值如果为 2, 表示串口中有数据需要读取。

(7) OnComm 事件。当串口中有事件发生时触发 OnComm 事件。语法如下:

```
void OnComm();
```

## 实现过程

本例分为客户端和服务端两个程序, 实现过程如下。

服务器端程序实现过程。

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加文本编辑、IP 地址控件, 并导入 ActiveX 控件 CMSComm。
- (3) 处理“打开串口”按钮的单击事件, 打开串口。

```
void CServerDlg::OnOpenport()
{
 UpdateData(TRUE);
 m_Comm.SetCommPort(m_Port+1); //设置串口
 m_Comm.SetRTThreshold(1); //设置收到多少个字符后触发OnComm事件
 m_Comm.SetPortOpen(TRUE); //打开串口
}
```

- (4) 处理“关机”按钮的单击事件, 关闭计算机。

```
void CServerDlg::OnExitwindows()
{
 CString str="关机";
 m_Comm.SetOutput((COleVariant)str);
 Sleep(1000); //延时1秒
}
```

客户端程序实现过程。

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加组合框控件、按钮控件, 并导入 ActiveX 控件 CMSComm。
- (3) 处理 CMSComm 控件的 OnComm 事件, 当接收到关机命令后, 执行关机。

```
void CRemoteCloseDlg::OnOnCommMscomm1()
{
 int resEvent;
 resEvent = m_Comm.GetCommEvent();
 switch (resEvent)
 {
 case 2: //接收数据
 {
 if (IsSended) //开始接收数据
 {
 VARIANT data;
 data = m_Comm.GetInput();

 CString str;
 str = data.bstrVal;
 IsSended = FALSE;
 }
 }
 }
}
```





```
if (str=="关机")
 OnOK();
```

```
break;
```

### 举一反三

根据本实例,读者可以:

通过短信远程关闭计算机。

## 11.2 加密狗和加密锁

在一些商业软件中,为了防止盗版,经常使用加密狗进行加密。本节将通过几个实例介绍如何利用加密狗进行程序加密。

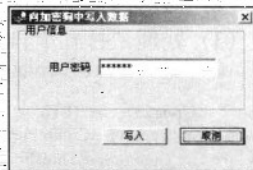
### 实例 323 将密码写入加密狗

本实例是一个人性化的实例

实例位置: 光盘\mingrisoft\11\323

#### 实例说明

在使用加密狗时,首先需要向加密狗中写入数据。例如,将密码写入加密狗。该如何实现呢?运行本例,在“用户密码”编辑框中设置密码后,单击“写入”按钮,即可将设置的密码写入加密狗。实例运行结果如图 11.3 所示。



#### 技术要点

在购买加密狗时,通常会附带有开发手册和一张光盘。开发手册中介绍了加密狗的使用方法和开发资料。本例使用赛孚耐信息技术有限公司的加密狗产品,它提供了使用 API 接口函数完成解密功能。下面介绍有关加密狗的 API 函数。

(1) WriteDog 函数。该函数用于向加密狗中写入数据。语法如下:

```
unsigned long WriteDog(void);
```

返回值: 如果执行成功,返回值为 0。

许多读者会对该函数产生疑问,向加密狗中写入的数据如何传递?它是通过全局变量实现的。为了增加加密狗的安全性,厂商定义了一个 C 语言标准加密模块 Rgdlw32v.obj。该模块包含了加密狗的 API 函数以及 3 个全局变量 DogBytes、DogAddr 和 DogData。代码如下:

```
short int DogBytes,DogAddr;
void * DogData;
```

其中 DogBytes 确定向加密狗中写入数据的长度。DogAddr 确定向加密狗中写入数据的起始位置。DogData 确定向加密狗中写入的数据。WriteDog 函数应用示例:

```
DogAddr = 10; //设置起始地址
DogBytes = m_Data.GetLength(); //设置数据的长度
DogData = m_Data.GetBuffer(0); //设置写入的数据
if (WriteDog()==0)
```

```
 MessageBox("数据写入成功");
```

(2) ReadDog 函数。该函数用于读取加密狗中的数据。语法如下:

```
unsigned long ReadDog(void);
```

返回值: 如果执行成功,返回值为 0。

ReadDog 函数应用示例:

```
DogAddr = 10; //设置起始地址
DogBytes = 4; //设置数据的长度
```

```
DogData = m_Text.GetBuffer(4); //设置写入的数据
```

```
if (ReadDog()==0)
{
 MessageBox("读取成功成功");
}
```

注意：在使用上述函数前，需要向工程中添加 Rgdlw32v.obj 和 SOFTDOG.h 文件。这两个文件可以在开发商的示例中找到。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加静态文本控件、文本编辑控件、按钮控件。
- (3) 向工程中添加 Rgdlw32v.obj 和 SOFTDOG.h 文件。
- (4) 处理“写入”按钮的单击事件，向加密狗中写入数据。

```
void CDOGDlg::OnOK()
{
 UpdateData();
 if (m_Data.IsEmpty())
 return;
 DogAddr = 10; //设置起始地址
 DogBytes = m_Data.GetLength(); //设置数据的长度
 DogData = m_Data.GetBuffer(0); //设置写入的数据

 if (WriteDog()==0)
 MessageBox("数据写入成功");
}
```

## 举一反三

根据本实例，读者可以：

- 利用加密狗进行程序加密。

## 实例 324 使用加密狗进行身份验证

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\11\324

## 实例说明

在开发程序时，对于一些机密的数据，需要很好的保护。例如，对于用户的密码，如果从数据库中验证用户密码，很容易被其他人发现。本例实现了利用加密狗进行身份验证。实例运行结果如图 11.4 所示。

## 技术要点

本例的关键是从加密狗中读取数据，可以使用 ReadDog 函数实现。有关该函数的介绍请参考实例“将密码写入加密狗”中的“技术要点”部分。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加静态文本控件、文本编辑控件、按钮控件。
- (3) 向工程中添加 Rgdlw32v.obj 和 SOFTDOG.h 文件。
- (4) 处理“登录”按钮的单击事件，向加密狗中写入数据。

```
void CDOGDlg::OnOK()
{
 UpdateData();
```

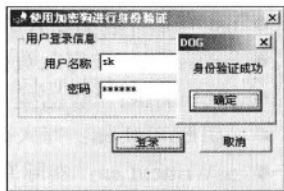


图 11.4 使用加密狗进行身份验证

```

if (m_Data.IsEmpty())
 return;
DogAddr = 10; //设置起始地址
DogBytes = 6; //设置数据的长度
DogData = m_Text.GetBuffer(6); //设置写入的数据

if (ReadDog() == 0)
{
 if (m_Text == m_Data)
 MessageBox("身份验证成功");
 else
 MessageBox("身份验证失败");
}
}

```

## 举一反三

根据本实例，读者可以：

- 利用加密狗设计加密软件。

## 实例 325 将数据写入加密锁

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\11\325

## 实例说明

加密锁是现阶段新型的数据加密设备，通常应用于软件的加密。通过将软件注册信息写入加密锁以维护正版软件市场。本实例实现了将数据写入加密锁，实例运行结果如图 11.5 所示。

## 技术要点

向加密锁中写数据主要是通过 EleT2Write 函数实现的，此函数定义在加密锁生产厂商所提供的动态链接库中。此函数的定义如下：

```

unsigned long EleT2Write(
 unsigned short usOffset,
 char* pcPassword,
 unsigned char *pucInbuffer,
 unsigned short usInbufferLen,
 unsigned short*usWrittenLen
);

```

参数说明：

- usOffset：数据写入的初始位置。
- pcPassword：写入数据时所需要的密码。
- pucInbuffer：数据缓存区指针。
- usInbufferLen：写入数据的长度。
- usWrittenLen：实际写入数据的长度。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加两个静态文本控件，设置 Caption 属性为“用户名”和“密码”；添加两个文本编辑框控件；添加一个按钮，设置 Caption 属性为“写入”。

(3) 主要实现代码如下：

```

void CWriteLockDlg::OnButwrite()
{
 // TODO: Add your control notification handler code here
 UpdateData();
 char* pcPassword="0000000000000000";
}

```

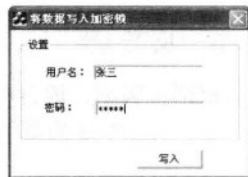


图 11.5 将数据写入加密锁

```

char* buf;
strcpy(buf,m_Data);
unsigned short usWrittenLen;
long ret = EleT2Write(0,pcPassword,(unsigned char *)buf,
 6,&usWrittenLen);
switch(ret)
{
case ELE_T2_SUCCESS:
 MessageBox("函数执行成功");
 break;
case ELE_T2_NO_MORE_DEVICE :
 MessageBox("没有找到相应的模板设备");
 break;
case ELE_T2_INVALID_PASSWORD :
 MessageBox("无效的密码");
 break;
case ELE_T2_INSUFFICIENT_BUFFER:
 MessageBox("缓冲区不足");
 break;
case ELE_T2_BEYOND_DATA_SIZE:
 MessageBox("读写数据区越界");
 break;
}
}

```

//写入数据  
//判断是否成功

### 举一反三

根据本实例，读者可以：

- 利用加密锁设计加密软件。

## 实例 326 使用加密锁进行软件注册

本实例是一个人性化的实例

实例位置：光盘\mingrsoft\11\326

### 实例说明

在一些商业软件中，为了防止盗版，在销售时经常会附赠一个加密锁，这样可以大大的增强安全性，本例就是通过 Visual C++ 来操作加密锁，从而在加密锁中进行写入和读取操作。运行本实例，在编辑框中写入注册码，单击“注册”按钮，程序将读取加密锁中的数据与用户输入的数据进行比较。实例运行结果如图 11.6 所示。

### 技术要点

本示例中实现使用加密锁进行软件加密时，主要用 EleT2Write 函数和 EleT2Read 函数，在使用这些函数前，需要在工程导入 elet2.h、elet2.lib、elet2.dll 文件。这些个文件可以在开发商的示例中找到。下面对本实例中用到的关键技术进行详细讲解。

(1) EleT2Write 函数。EleT2Write 函数用于向加密锁模板中写入数据。该方法的语法格式如下：

```

unsigned long EleT2Write(unsigned short usOffset, char* pcPassword, unsigned char *puclnbuffer, unsigned short usInbufferLen, unsigned short usWrittenLen);

```

参数说明：



图 11.6 使用加密锁进行软件注册



- usOffset: 偏移量, 从数据区起始位置偏移多少开始写入。
- pcPassword: 写口令。
- pucInbuffer: 存放数据的缓冲区。
- usInbufferLen: 写入数据的长度。
- usWrittenLen: 实际写入到数据区的长度。

(2) EleT2Read 函数。EleT2Read 函数用于设置读取加密锁中的数据。其语法格式如下:

```
unsigned long EleT2Read(unsigned short usOffset, unsigned char * pucOutbuffer, unsigned short usOutbufferLen, unsigned short usReadLen);
```

参数说明:

- usOffset: 偏移量, 从数据区起始位置偏移多少开始读取。
- pucOutbuffer: 存放数据的缓冲区。
- usOutbufferLen: 存放数据的长度。
- usReadLen: 实际读取的数据区的长度。

## 实现过程

- (1) 新建一个基于对话框的应用程序, 将其窗体标题改为“使用加密锁进行软件加密”。
- (2) 向工程中导入两个 BMP 位图资源。
- (3) 向对话框中添加一个图片控件、一个编辑框控件和一个按钮控件。
- (4) 将 elet2.h、elet2.lib、elet2.dll 文件导入到工程中。
- (5) 在主窗口初始化时, 为“注册”按钮设置显示图片, 向加密锁中写入字符串, 其实现

代码如下:

```
m_Login.SetBitmap(LoadBitmap(AfxGetInstanceHandle(), MAKEINTRESOURCE(IDB_BITMAP2))); //设置位图
char* pcPassword="0000000000000000"; //设置口令
char* buf="123456"; //设置写入字符串
unsigned short usWrittenLen;
long ret = EleT2Write(0, pcPassword, (unsigned char *)buf, 6, &usWrittenLen); //写入数据
switch(ret) //判断是否成功
{
case ELE_T2_SUCCESS: //函数执行成功
 MessageBox("函数执行成功");
 break;
case ELE_T2_NO_MORE_DEVICE: //没有找到相应的模板设备
 MessageBox("没有找到相应的模板设备");
 break;
case ELE_T2_INVALID_PASSWORD: //无效的密码
 MessageBox("无效的密码");
 break;
case ELE_T2_INSUFFICIENT_BUFFER: //缓冲区不足
 MessageBox("缓冲区不足");
 break;
case ELE_T2_BEYOND_DATA_SIZE: //读写数据区越界
 MessageBox("读写数据区越界");
 break;
}
```

(6) 处理“注册”按钮的单击事件, 在该事件的处理函数中调用 EleT2Read 函数读取加密锁中的数据, 并判断读取的数据和用户的数据是否相同, 其实现代码如下:

```
void CEncryptLockDlg::OnButlogin()
{
 unsigned short usReadLen; //保存读取数据长度
 CString strtext;Data; //声明字符串变量
 char buf[6]; //声明字符数组
 long ret = EleT2Read(0, (unsigned char *)buf, 6, &usReadLen); //读取加密锁中数据
 switch(ret) //判断是否发生错误
 {
case ELE_T2_NO_MORE_DEVICE: //没有找到相应的模板设备
 MessageBox("没有找到相应的模板设备");
 break;
case ELE_T2_INVALID_PASSWORD: //无效的密码
```

```

 MessageBox("无效的密码");
 break;
 case ELE_T2_INSUFFICIENT_BUFFER: //缓冲区不足
 MessageBox("缓冲区不足");
 break;
 case ELE_T2_BEYOND_DATA_SIZE: //读写数据区越界
 MessageBox("读写数据区越界");
 break;
 }
 buf[6]='\0'; //设置字符串结束符
 strtext = buf;
 m_Data.GetWindowText(Data); //获得用户输入数据
 if(ret != ELE_T2_SUCCESS || strtext != Data) //判断读取数据是否成功以及输入数据是否相同
 {
 MessageBox("请插入指定的加密锁");
 CDialog::OnCancel();
 }
 else //读取数据成功，输入数据与读取数据相同
 {
 MessageBox("注册成功");
 }
}

```

### 举一反三

根据本实例，读者可以：

- 利用加密狗设计加密软件。

## 11.3 IC 卡、ID 卡应用

IC 卡也被称之为智能卡，具有存储数据的能力。它具有抗干扰、无磨损、寿命长等特点，因此被广泛地应用于各个领域。本节通过几个实例介绍 IC 卡应用程序的开发。

### 实例 327 向 IC 卡中写入数据

本实例是一个人性化的实例

实例位置：光盘\mingrisoft\11\327

### 实例说明

IC 卡是携带应用信息和数据的存储媒体，一张空的 IC 卡是不能立即使用的，需要用户向 IC 卡中写入数据。本例实现了向 IC 卡中写入数据的功能。效果如图 11.7 所示。

### 技术要点

本例使用的是深圳明华生产的明华 IC 卡读卡器(推推式)。IC 卡采用西门子 SLE4442 型号，该型号的 IC 卡在写入数据前，需要进行密码核对，如果密码 3 次核对错误，IC 卡将自动锁死，不能够再向其写入数据，也不能够进行密码核对。西门子 SLE4442 型号 IC 卡的初始密码为“ffff”。在 IC 卡读卡器的驱动程序中包含了 Mwic\_32.dll 动态链接库，有关操作 IC 卡的函数都封装在 Mwic\_32.dll 动态链接库中。下面介绍与 IC 卡有关的重要函数。

(1) auto\_init 函数。该函数用于初始化 IC 卡读卡器。语法如下：

```
int auto_init(int port,unsigned long baud);
```

参数说明：

- port：标识端口号，Com1 对应的端口号为 0，Com2 对应的端口号为 1，依此类推。

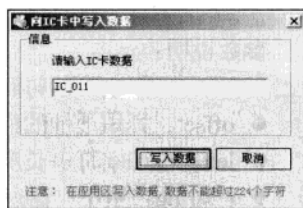


图 11.7 向 IC 卡中写入数据

● baud: 标识波特率。

● 返回值: 如果初始化成功, 返回值是 IC 卡设备句柄, 如果初始化失败, 返回值小于零。

(2) setsc\_md 函数。该函数用于设置设备密码模式。语法如下:

```
int setsc_md(int icdev, int mode);
```

参数说明:

● icdev: 标识设备句柄, 通常是 auto\_init 函数的返回值。

● mode: 标识设备密码模式, 如果为 0, 设备密码有效, 设备在加电时必须验证设备密码才能对设备进行操作; 如果为 1, 设备密码无效。

● 返回值: 如果函数执行成功返回值为零, 否则小于零。

(3) get\_status 函数。该函数用于获取设备的当前状态。语法如下:

```
int get_status(int icdev, int *state);
```

参数说明:

● icdev: 标识设备句柄, 通常是 auto\_init 函数的返回值。

● state: 用于接收函数返回的结果, 如果为 0 表示读卡器中无卡; 为 1, 表示读卡器中有卡。

● 返回值: 如果函数执行成功返回值为零, 否则小于零。

(4) csc\_4442 函数。该函数用于核对 IC 卡密码。语法如下:

```
int csc_4442(int icdev, int len, unsigned char* p_string);
```

参数说明:

● icdev: 标识设备句柄, 通常是 auto\_init 函数的返回值。

● len: 标识密码长度, 其值为 3。

● p\_string: 标识设置的密码。

● 返回值: 如果函数执行成功返回值为零, 否则小于零。

(5) swr\_4442 函数。该函数用于向 IC 卡中写入数据。语法如下:

```
int swr_4442(int icdev, int offset, int len, unsigned char *w_string);
```

参数说明:

● icdev: 标识设备句柄, 通常是 auto\_init 函数的返回值。

● offset: 标识地址的偏移量, 范围是 0~255。

● len: 标识字符串长度。

● w\_string: 标识写入的数据。

(6) srd\_4442 函数。该函数用于读取 IC 卡中的数据。语法如下:

```
int srd_4442(int icdev, int offset, int len, unsigned char* r_string);
```

参数说明:

● icdev: 标识设备句柄, 通常是 auto\_init 函数的返回值。

● offset: 标识地址的偏移量, 范围是 0~255。

● len: 标识字符串长度。

● r\_string: 用于存储返回的数据。

(7) ic\_exit 函数。该函数用于关闭设备端口。语法如下:

```
int ic_exit(int icdev);
```

参数说明:

● icdev 标识设备句柄, 通常是 auto\_init 函数的返回值。

(8) dv\_beep 函数。该函数使读卡器嗡鸣。语法如下:

```
int dv_beep(int icdev, int time);
```

参数说明:

● icdev: 标识设备句柄, 通常是 auto\_init 函数的返回值。

● time: 标识嗡鸣持续的时间, 单位是 10ms。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加静态文本控件、文本编辑控件和按钮控件。
- (3) 将“Mwic\_32.dll”、“Mwic\_32.lib”和“Mwic\_32.h”这 3 个文件放置在工程文件目录下，用户可以在驱动程序目录下找到这些文件。

- (4) 在对话框初始化时初始读卡器，验证密码。

```
//初始化端口 COM1 9600
icdev= auto_init(0,9600);
if (icdev<0)
 MessageBox("端口初始化失败,请检查接口线是否连接正确.");

//设置密码模式,使设备密码无效,否则在设备加电时,必须核对密码才能进行后续操作
setsc_md(icdev,1);
__int16 rpass;
//获取IC卡状态
__int16 status = -1;

__int16 result =get_status(icdev,&status);

if (result<0)
{
 MessageBox("获取IC卡状态错误");
}
else if ((result==0)&&(status==0))
 MessageBox("请插入IC卡");

//西门子4442IC卡初试密码为“fmmf”
unsigned char pass[3]={0xff,0xff,0xff};
//验证密码
rpass =csc_4442(icdev,3,pass);
if (rpass<0)
{
 MessageBox("IC卡密码核对错误,只能读取数据");
}
}
```

- (5) 处理“写入数据”按钮的单击事件，向 IC 卡中写入数据。

```
void CICCCardDlg::OnOK()
{
 if (MessageBox("确实要写入数据吗?","提示",MB_YESNO)==IDYES)
 {
 CString str;
 m_ICData.GetWindowText(str);
 if (str.IsEmpty())
 {
 MessageBox("请输入数据");
 return;
 }
 if (str.GetLength()>224)
 {
 MessageBox("写入数据不能超过224个字符","提示",64);
 return;
 }

 __int16 result;

 //在IC卡的应用区中写入数据
 result =swr_4442(icdev,33,str.GetLength(),(unsigned char*)
str.GetBuffer(0));

 if (result==0)
 {
 MessageBox("数据写入成功.", "提示",64);
 }
 else
 {
 MessageBox("数据写入失败.", "提示",64);
 }
 }
}
```



## 举一反三

根据本实例,读者可以:

- 开发 IC 卡考勤系统。

## 实例 328 读取 IC 卡中的数据

本实例是一个非常实用的程序

实例位置: 光盘\mingrisoft\11\328

## 实例说明

在向 IC 卡中写入数据后,就可以读取 IC 卡中的数据了。运行本例,单击“读卡”按钮,会将 IC 卡中的数据显示在编辑框中。实例运行结果如图 11.8 所示。

## 技术要点

本例中主要调用 `srd_4442` 函数读取 IC 卡中的数据,详细介绍请参考实例“向 IC 卡中写入数据”中的“技术要点”部分。

## 实现过程

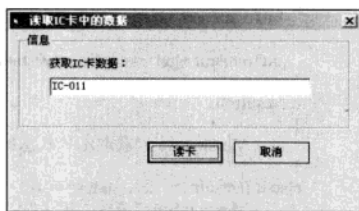


图 11.8 读取 IC 卡中的数据

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加静态文本控件、文本编辑控件和按钮控件。
- (3) 将“Mwic\_32.dll”、“Mwic\_32.lib”和“Mwic\_32.h”3个文件放置在工程文件目录下,用户可以在驱动程序目录下找到这些文件。
- (4) 在对话框初始化时初始读卡器。

```
//初始化端口 COM1 9600
icdev= auto_init(0,9600);
if (icdev<0)
 MessageBox("端口初始化失败,请检查接口线是否连接正确.");

//设置密码模式,使设备密码无效,否则在设备加电时,必须核对密码才能进行后续操作
setsc_md(icdev,1);
//获取IC卡状态
__int16 status = -1;

__int16 result =get_status(icdev,&status);

if (result<0)
{
 MessageBox("获取IC卡状态错误");
}
else if ((result==0)&&(status==0))
 MessageBox("请插入IC卡");
```

- (5) 处理“读卡”按钮的单击事件,读取 IC 卡数据。

```
void CReadCardDlg::OnRead()
{
 __int16 result;
 unsigned char data[224];
 result = srd_4442(icdev,33,223,data);
 //嘀鸣
 dv_beep(icdev,20);
 if (result<0)
 MessageBox("数据读取失败");
 else
 {
 int i =0;
 for (i= 0; i<224;i++)
 {
```

```

 if (data[i]==255)
 break;
 }
 unsigned char* pArray = new unsigned char[i+1];
 memset(pArray,0,i+1);
 memcpy(pArray,data,i);
 pArray[i]=0;
 m_Data.SetWindowText((char*)pArray);
 delete pArray;
}
}

```

### 举一反三

根据本实例，读者可以：  
● 开发 IC 卡电话系统。

## 实例 329 利用 IC 卡制作考勤程序

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrsoft\11\329

### 实例说明

随着科学技术的不断发展，IC 卡应用越来越广泛。如银行卡、公交 IC 卡、借书卡等。本例将利用 IC 卡制作一个考勤系统。实例运行结果如图 11.9 所示。

### 技术要点

有关 IC 卡的操作函数请参考实例“向 IC 卡中写入数据”中的“技术要点”部分。下面主要介绍在“考勤窗口”中如何判断有 IC 卡插入，并能够立即读取 IC 卡中的信息。

由于程序中需要时时检查读卡器中是否有 IC 卡，因此，可以设置一个计时器，在对话框的 OnTimer 事件中进行判断。如果有 IC 卡，读取 IC 卡中的信息，并根据 IC 卡中的信息从数据表中查询员工信息。具体代码请参考实现过程。

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 创建一个对话框，本例为 CReFreshCard，作为考勤窗口。
- (3) 在 CReFreshCard 对话框中添加文本编辑控件、组合框控件和静态文本控件。
- (4) 将“Mwic\_32.dll”“Mwic\_32.lib”和“Mwic\_32.h”这 3 个文件放置在工程文件目录下，用户可以在驱动程序目录下找到这些文件。
- (5) 处理 CReFreshCard 对话框的 WM\_TIMER 消息，判断读卡器中是否存在 IC 卡，如果存在，读取其信息。

```

void CReFreshCard::OnTimer(UINT nIDEvent)
{
 //获取IC卡状态
 _int16 status = -1;
 _int16 result =get_status(m_icdev,&status);

 setsc_md(m_icdev,1);
}

```

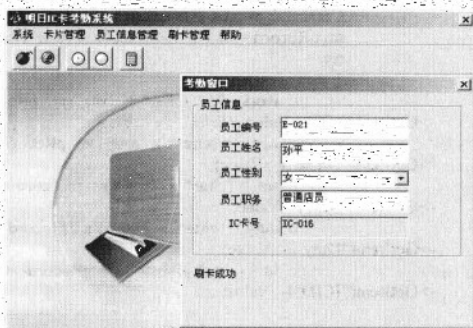


图 11.9 利用 IC 卡制作考勤程序

```

if (result<0)
{
 return;
}
else if ((result==0)&&(status==0))
{
 m_IsCard = FALSE;
 m_HINT.SetWindowText("请插入IC卡");
 return;
}

if (m_IsCard)
 return;

//读取卡中的数据
unsigned char data[224];
result = srds_4442(m_icdev,33,223,data);
//嘀嗒
dw_bEEP(m_icdev,20);
if (result==0)
{
 int i=0;
 for (i= 0; i<224;i++)
 {
 if (data[i]==255)
 break;
 }
 unsigned char* pArray = new unsigned char[i+1];
 memset(pArray,0,i+1);
 memcpy(pArray,data,i);
 pArray[i]= 0;

 CString sql;
 sql.Format("select * from worker where icid = '%s'",pArray);
 delete pArray;

 CString c_workerid,c_name,c_sex,c_duty,c_id;
 m_pRecord = m_pCon->Execute((_bstr_t)sql,NULL,-1);//执行SQL语句
 try
 {
 c_workerid = (char*)(_bstr_t)m_pRecord->GetFields()
->GetItem("ID")>Value;//获取字段数据
 c_name = (char*)(_bstr_t)m_pRecord->GetFields()
->GetItem("Name")>Value;
 c_sex = (char*)(_bstr_t)m_pRecord->GetFields()
->GetItem("Sex")>Value;
 c_duty = (char*)(_bstr_t)m_pRecord->GetFields()
->GetItem("Duty")>Value;
 c_id = (char*)(_bstr_t)m_pRecord->GetFields()
->GetItem("ICID")>Value;

 m_WorkID.SetWindowText(c_workerid);
 m_Name.SetWindowText(c_name);
 if (c_sex=="男")
 m_Sex.SetCurSel(0);
 else
 m_Sex.SetCurSel(1);

 m_Duty.SetWindowText(c_duty);
 m_ID.SetWindowText(c_id);

 m_IsCard = TRUE;

 sql.Format("insert into record values ('%s','%s')",
c_id,CTime::GetCurrentTime().Format("%Y-%m-%d %H:%M:%S"));

 try
 {
 m_pCon->Execute((_bstr_t)sql,NULL,-1);//执行SQL语句
 m_HINT.SetWindowText("刷卡成功");
 }
 catch(...)
 {

```

```

 m_HINT.SetWindowText("数据记录失败,请重新插入IC卡");
 }
}
catch(...)
{
 m_WorkID.SetWindowText("");
 m_Name.SetWindowText("");
 m_Sex.SetWindowText("");
 m_Duty.SetWindowText("");
 m_ID.SetWindowText("");
 m_HINT.SetWindowText("提示: 该卡不存在用户");
 dv_beep(m_icdev,10);
 dv_beep(m_icdev,10);
 m_IsCard =TRUE;
}
}
CDialog::OnTimer(nIDEvent);
}

```

### 举一反三

根据本实例，读者可以：

- 开发公交 IC 卡系统。

## 实例 330 使用 ID 卡制作考勤程序

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\11\330

### 实例说明

ID 卡是现在非常流行的卡，该卡与磁卡有着很大的区别。其最大的区别就在于磁卡在使用一段时间后就会出现消磁的现象，而 ID 卡不会消磁，所以适用性更强、更容易携带。本实例可利用 ID 卡实现员工的考勤，实例运行结果如图 11.10 所示。

### 技术要点

ID 卡是与计算机的键盘口串行连接一个硬件设备，该设备主要是对计算机的当前窗口发送指定位数的字符串 ID 值，最后再发送一个键盘回车消息。在本实例中为了在任何时候都可以获取 ID 卡中的值，使用全局键盘钩子来截获读卡设备发送的键盘消息。这个全局钩子的类型为 WH\_KEYBOARD\_LL，并使用 SetWindowsHookEx 函数来安装钩子，当钩子使用完后再通过 UnhookWindowsHookEx 函数来卸载钩子。

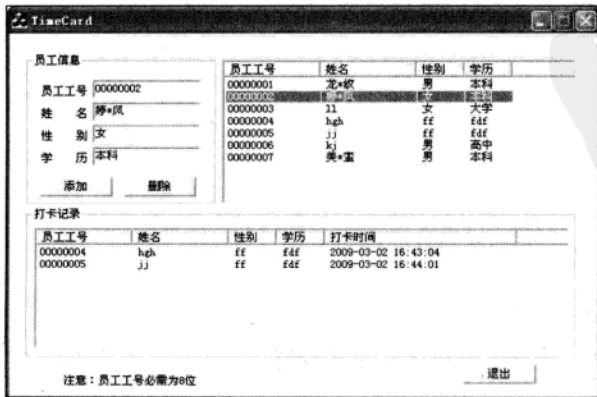


图 11.10 使用 ID 卡制作考勤程序



在 WH\_KEYBOARD\_LL 类型的钩子使用时需要定义一个结构来获取键盘信息, 该结构类型的定义如下:

```
typedef struct tagKBDLLHOOKSTRUCT {
 DWORD vkCode;
 DWORD scanCode;
 DWORD flags;
 DWORD time;
 DWORD dwExtraInfo;
} KBDLLHOOKSTRUCT, FAR *LPKBDLLHOOKSTRUCT, *PKBDLLHOOKSTRUCT;
```

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框中添加 4 个静态文本框, 设置其 Caption 属性为“员工工号”、“姓名”、“性别”和“年龄”; 添加 2 个列表控件, 设置 View 属性为 Report; 添加 3 个按钮控件, 设置 Caption 属性为“添加”、“删除”和“退出”。

(3) 系统键盘钩子的加载与卸载, 实现代码如下:

```
BOOL installhook()//安装钩子
{
 index = 0;
 hkb=SetWindowsHookEx(WH_KEYBOARD_LL,(HOOKPROC)KeyboardProc,::AfxGetApp()->m_hInstance,0);
 return TRUE;
}

BOOL UnHook()//卸载钩子
{
 BOOL unhooked = UnhookWindowsHookEx(hkb);
 return unhooked;
}
```

(4) 钩子回调函数的实现, 获取 ID 卡号码。实现代码如下:

```
LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam)
{
 KBDLLHOOKSTRUCT *pkh = (KBDLLHOOKSTRUCT *) lParam;
 if(HC_ACTION==nCode)
 {
 if (wParam == WM_KEYDOWN)//键盘按下
 {
 if (index == 8 &&
 ((pkh->vkCode >= 48 && pkh->vkCode <= 57)
 || (pkh->vkCode >= VK_NUMPAD0 && pkh->vkCode <= VK_NUMPAD9)))
 {
 for (int i = 0; i < 7; i++)
 str[i] = str[i + 1];
 index = 7;
 }

 switch (pkh->vkCode)//向缓冲区中存入数据
 {
 case 48: str[index++] = '0'; break;
 case 49: str[index++] = '1'; break;
 case 50: str[index++] = '2'; break;
 case 51: str[index++] = '3'; break;
 case 52: str[index++] = '4'; break;
 case 53: str[index++] = '5'; break;
 case 54: str[index++] = '6'; break;
 case 55: str[index++] = '7'; break;
 case 56: str[index++] = '8'; break;
 case 57: str[index++] = '9'; break;
 case VK_NUMPAD0: str[index++] = '0'; break;
 case VK_NUMPAD1: str[index++] = '1'; break;
 case VK_NUMPAD2: str[index++] = '2'; break;
 case VK_NUMPAD3: str[index++] = '3'; break;
 case VK_NUMPAD4: str[index++] = '4'; break;
 case VK_NUMPAD5: str[index++] = '5'; break;
 case VK_NUMPAD6: str[index++] = '6'; break;
 case VK_NUMPAD7: str[index++] = '7'; break;
 case VK_NUMPAD8: str[index++] = '8'; break;
```

```

 case VK_NUMPAD9: str[index++] = '9'; break;
 }
 if (pkh->vkCode == 13) //回车键
 {
 if (Ghookpro != NULL)
 {
 str[8] = '\0';
 CString tempstr;
 tempstr.Format("select* from picture where 卡号 = '%s'", str);
 _RecordsetPtr RecordSet;
 RecordSet.CreateInstance(__uuidof(Recordset));
 RecordSet->Open((_variant_t)(tempstr), pConnection.GetInterfacePtr(), adOpenKeyset, adLockOptimistic, adCmdText);
 if (RecordSet->RecordCount == 1) //查询记录是否存在
 Ghookpro(str);
 }
 }
}
LRESULT RetVal = CallNextHookEx(hkb, nCode, wParam, lParam);
return RetVal;
}
}

```

(5) 获取 ID 卡号后向数据表中插入考勤数据，实现代码如下：

```

void Callhookpro(char * str)
{
 CTime ti = CTime::GetCurrentTime();
 CString Date, Time, CommandText;
 Date.Format("%d-%d-%d", ti.GetYear(), ti.GetMonth(), ti.GetDay());
 Time.Format("%d:%d:%d", ti.GetHour(), ti.GetMinute(), ti.GetSecond());
 CommandText.Format("insert into timeCard(员工工号, 打卡日期, 打卡时间)\n
 values('%s', '%s', '%s')", str, Date, Time);

 _CommandPtr m_pCommand;
 m_pCommand.CreateInstance(__uuidof(Command));
 m_pCommand->ActiveConnection = dlg->m_pConnection;
 m_pCommand->CommandText = (_bstr_t)CommandText;
 m_pCommand->Execute(NULL, NULL, adCmdText); //执行SQL语句
 dlg->UpdateCardList(); //更新人员列表
}

```

(6) 在窗体初始化时显示人员信息列表，实现代码如下：

```

void CTimeCardDlg::InitPersonList()
{
 m_pRecordset.CreateInstance(__uuidof(Recordset)); //创建数据集
 m_pRecordset->Open("select* from\n
 picture", m_pConnection.GetInterfacePtr(), adOpenKeyset, adLockOptimistic, adCmdText); //打开数据集
 int m = m_pRecordset->GetRecordCount(); //获取查询记录数量

 CString temp;
 int index = 0;
 m_personlist.DeleteAllItems();
 while(!m_pRecordset->adoEOF) //循环数据集
 {
 temp = (char*)(_bstr_t)m_pRecordset->GetCollect("卡号");
 m_personlist.InsertItem(index, temp);

 m_personlist.SetItemText(index, 1, (char*)(_bstr_t)m_pRecordset->GetCollect("姓名"));
 m_personlist.SetItemText(index, 2, (char*)(_bstr_t)m_pRecordset->GetCollect("性别"));
 m_personlist.SetItemText(index, 3, (char*)(_bstr_t)m_pRecordset->GetCollect("学历"));
 index++;
 m_pRecordset->MoveNext();
 }
}

```

(7) 更新考勤数据表中的数据，实现代码如下：

```

void CTimeCardDlg::UpdateCardList()
{
 m_cardlist.DeleteAllItems();
 m_pRecordset.CreateInstance(__uuidof(Recordset)); //创建数据集
 m_pRecordset->Open("select a.*, b.打卡日期, b.打卡时间 from picture a, timecard b\n
 where a.卡号 = b.员工工号 order by a.卡号", m_pConnection.GetInterfacePtr(), adOpenKeyset, adLockOptimistic, adCmdText); //执行查询语句
 int count = m_pRecordset->GetRecordCount(); //获取数据集记录数量
 for (int i = 0; i < count; i++) //循环数据集
 {

```



```

CString time;
time.Format("%s %s", (char*)(_bstr_t)m_pRecordset->GetCollect("打卡日期"),
(char*)(_bstr_t)m_pRecordset->GetCollect("打卡时间"));
m_cardlist.InsertItem(i, (char*)(_bstr_t)m_pRecordset->GetCollect("卡号"));
m_cardlist.SetItemText(i, 1, (char*)(_bstr_t)m_pRecordset->GetCollect("姓名"));
m_cardlist.SetItemText(i, 2, (char*)(_bstr_t)m_pRecordset->GetCollect("性别"));
m_cardlist.SetItemText(i, 3, (char*)(_bstr_t)m_pRecordset->GetCollect("学历"));
m_cardlist.SetItemText(i, 4, time);
m_pRecordset->MoveNext();
}
}

```

### 举一反三

根据本实例，读者可以：

- 获取 ID 卡刷卡次数。

## 11.4 监 控

随着科学技术的不断发展，如今的许多商场和小区都配备有监控系统，起到安全的作用。本节将介绍监控软件的开发技术。

### 实例 331

### 利用简易摄像头编写监控程序

这是一个可以提高基础性能的实例

实例位置：光盘\mingrisoft\11\331

### 实例说明

摄像头具有监控的功能。那么如何在程序中显示摄像头捕获的信息呢？本例实现了该功能。运行程序，在窗口中将显示摄像头捕获的信息。效果如图 11.11 所示。

### 技术要点

本例主要使用 VFW 技术实现视频的捕捉。VFW 是 Microsoft 公司推出的一个数字视频软件包，它能使应用程序数字化并播放从传统模拟视频源得到的视频剪辑。VFW 主要由一组动态库构成，在安装操作系统时会自动安装。为了支持 VFW，Visual C++ 提供了 vfw32.lib、msacm32.lib、winmm.lib 库文件。在这些库文件中提供了多个函数和宏用于视频应用程序的开发。下面介绍主要的视频函数。



图 11.11 利用简易摄像头编写监控程序

(1) capCreateCaptureWindow 函数。该函数用于创建一个视频捕捉窗口。语法如下：

```

HWND VFWAPI capCreateCaptureWindow(LPCSTR lpszWindowName, DWORD dwStyle, int x, int y, int nWidth, int nHeight, HWND hWnd, int nID);

```

参数说明：

- lpszWindowName：标识窗口的名称。
- dwStyle：标识窗口风格。
- x、y：标识窗口的左上角坐标。
- nWidth、nHeight：标识窗口的宽度和高度。
- hWnd：标识父窗口句柄。
- nID：标识窗口 ID。
- 返回值：视频捕捉窗口句柄。

(2) capDriverConnect 宏。该宏主要将视频捕捉窗口连接到视频驱动程序上。语法如下：

BOOL capDriverConnect(hwnd, iIndex);

参数说明:

- hwnd: 标识视频捕捉窗口句柄。
- iIndex: 标识驱动程序, 范围是 0~9。

(3) capPreviewRate 宏。该宏在预览模式下设置帧的显速率。语法如下:

BOOL capPreviewRate(hwnd, wMS);

参数说明:


- hwnd: 标识视频捕捉窗口句柄。
- wMS: 标识速率, 单位是毫秒。

(4) capPreview 宏。该宏用于激活或禁止预览模式。在预览模式下, 视频设备捕捉的数据以帧的形式传递到系统内存中, 然后通过 GDI 函数显示在视频捕捉窗口中。语法如下:

BOOL capPreview(hwnd, f);

参数说明:

- hwnd: 标识视频捕捉窗口句柄。
- f: 标识激活或禁止预览模式。

 注意: 为了使用上述函数和宏, 程序中需要引用 vfw.h 头文件, 连接 vfw32.lib 库文件。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框类的头文件中引用 vfw.h 头文件, 连接 vfw32.lib 库文件。
- (3) 在对话框类中定义一个静态方法, 作为线程函数。

```
static UINT ThreadFun(LPVOID lpParam);
```

- (4) 在对话框初始化时开始一个线程, 连接设备驱动程序。

```
AfxBeginThread(ThreadFun,(void*)this); //开始一个线程
```

```
::WaitForSingleObject(m_Event,INFINITE); //等待线程函数的执行
```

```
if (capDriverConnect(m_hVideoWnd,0)) //连接驱动程序
```

```
{
 ::SetParent(m_hVideoWnd,*this); //设置父窗口
 ::SetWindowLong(m_hVideoWnd,GWL_STYLE,WS_CHILD);
 ::SetWindowPos(m_hVideoWnd,NULL,10,10,
 300,300,SWP_NOZORDER);
```

```
::ShowWindow(m_hVideoWnd,SW_SHOW);
capPreviewRate(m_hVideoWnd,30); //设置预览速率
capPreview(m_hVideoWnd,true); //开始预览
}
```

- (5) 在线程函数中创建视频捕捉窗口。

//定义线程函数

```
UINT CCaptureAppDlg::ThreadFun(LPVOID lpParam)
```

```
{
 CCaptureAppDlg* pTempDlg = (CCaptureAppDlg*)lpParam;
 if (pTempDlg!= NULL)
 {
 HWND hwnd= capCreateCaptureWindow(NULL,WS_POPUP,0,0,200,300
 ,pTempDlg->m_hWnd,0); //创建捕获窗口
 pTempDlg->m_hVideoWnd = hwnd;
 pTempDlg->m_Event.SetEvent();

 MSG msg;
 while(GetMessage(&msg,NULL,0,0))
 {
 TranslateMessage(&msg);
 DispatchMessage(&msg);
 }
 return msg.wParam;
 }
 return 0;
}
```



## 举一反三

根据本实例,读者可以:

- 开发视频监控程序。

## 实例 332 编写监控录像程序

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\11\332

## 实例说明

在开发视频应用程序时,监控录像是必不可少的。它能够将在摄像头捕捉的信息保存成文件存储在磁盘中。用户可以通过播放这些文件来查看信息。本例实现了监控录像的功能。效果如图 11.12 所示。

## 技术要点

在 VFW 技术中,提供了 `capFileSetCaptureFile` 宏和 `capCaptureSequence` 宏用于将监控设备捕捉的信息存成文件。

- (1) `capFileSetCaptureFile` 宏。该宏用于设置视频捕捉的文件名称。语法如下:

```
BOOL capFileSetCaptureFile(HWND, szName);
```

参数说明:

- `hwnd`: 标识视频捕捉窗口句柄。
- `szName`: 标识文件名称。

- (2) `capCaptureSequence` 宏。该宏用于初始化视频流,捕捉视频信息到文件。语法如下:

```
BOOL capCaptureSequence(HWND);
```

参数说明:

- `hwnd`: 标识视频捕捉窗口句柄。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框类的头文件中引用 `vfw.h` 头文件,连接 `vfw32.lib` 库文件。
- (3) 在对话框类中定义一个静态方法,作为线程函数。

```
static UINT ThreadFun(LPVOID lpParam);
```

- (4) 在对话框初始化时开始一个线程,连接设备驱动程序。

```
m_pThread = AfxBeginThread(ThreadFun,(LPVOID)this); //开始线程
m_Event.ResetEvent();
::WaitForSingleObject(m_Event,INFINITE); //线程等待
if (m_hVideoWnd)
{
 if (capDriverConnect(m_hVideoWnd,0)==FALSE)
 {
 MessageBox("连接驱动程序出错");
 return;
 }
 //设置捕获设置
 ::SetParent(m_hVideoWnd,*this);
 ::SetWindowLong(m_hVideoWnd,GWL_STYLE,WS_CHILD);
 ::SetWindowPos(m_hVideoWnd,NULL,10,10,300,300,SWP_NOREDRAW);
 ::ShowWindow(m_hVideoWnd,SW_SHOW);
 capPreviewRate(m_hVideoWnd,10);
 capPreview(m_hVideoWnd,TRUE);
}
```

- (5) 在线程函数中创建视频捕捉窗口。

//线程执行函数

```
UINT CKinescodeDlg::ThreadFun(LPVOID wParam)
```



图 11.12 监控录像程序

```

CKinescodeDlg* temp = (CKinescodeDlg*)wParam;
temp->m_hVideoWnd = capCreateCaptureWindow("Capture",
WS_POPUP,10,10,20,20,*temp,0);
if (temp->m_hVideoWnd)
{
 temp->m_Event.SetEvent();
 MSG msg;
 while (GetMessage(&msg,temp->m_hVideoWnd,0,0))
 {
 TranslateMessage(&msg);
 DispatchMessage(&msg);
 }
 return msg.wParam;
}
return 0;
}

```

(6) 处理“录像”按钮的单击事件，开始捕捉视频信息到文件。

```

void CKinescodeDlg::OnKinescode()
{
 CString filename;
 CFileDialog FileDlg(FALSE,"avi");
 if (FileDlg.DoModal()==IDOK)
 {
 filename = FileDlg.GetPathName();
 filename.GetBufferSetLength(filename.GetLength()+2);
 filename.Insert(filename.GetLength(),'');
 filename.Insert(filename.GetLength()+1,'0');
 capFileSetCaptureFile(m_hVideoWnd,filename.GetBuffer(0));//设置录制文件名
 capCaptureSequence(m_hVideoWnd);
 m_IsKindscode = TRUE;
 }
}

```

### 举一反三

根据本实例，读者可以：

- 开发车库安全时时监控系统。

## 实例 333 远程视频监控系统

这是一个可以提高基础性能的实例

实例位置：光盘\mingrisoft\11\333

### 实例说明

网上的许多聊天软件都具有远程视频聊天的功能。只要双方安装有摄像头，就可以将彼此摄像头捕获的图像显示在对方的聊天软件中。本例实现了远程视频监控的功能，运行程序，当客户端与服务端建立连接后，服务器端就可以获得客户端摄像头捕捉的图片。效果如图 11.13、图 11.14 所示。



图 11.13 客户端程序



图 11.14 服务器端程序

### 技术要点

本例中实现远程视频监控是通过传递图片实现的。客户端时时获得摄像头捕捉的图像，将其存

成文件, 发送到服务器端。服务器端则定时接收数据, 并将接收的数据以图片的形式显示在窗口中。

在 VFW 技术中提供有 capSetCallbackOnFrame 宏, 用于在应用程序中设置预览模式的回调函数。这样当摄像头捕获到每一帧时, 就会调用指定的回调函数。用户可以在回调函数中调用 capFileSaveDIB 宏将帧保存成图片。最后通过记时器时将图片传递到服务器端。

示例如下:

```
//回调函数
void CaptureFun(LPVOID pParam)
{
 if (m_IsSending==FALSE)
 {
 m_IsSave=FALSE;
 capFileSaveDIB(hTemp,"c:\\a.bmp");//设置保存路径
 m_IsSave = TRUE;
 }
}

void CKinescodeDlg::OnStopkinescode()
{
 capSetCallbackOnFrame(m_hVideoWnd,CaptureFun);
 m_IsSend = TRUE;
 SetTimer(1,200,NULL);
}
```

## 实现过程

客户端应用程序设计步骤如下。

(1) 新建一个基于对话框的应用程序。

(2) 向对话框中添加按钮控件。

(3) 在对话框类的头文件中引用 vfw.h 头文件, 连接 vfw32.lib 库文件。

```
#include "vfw.h"
#pragma comment (lib,"vfw32.lib")
```

(4) 从 CSocket 类派生一个子类, 本例为 CClientSocket。在该类中定义一个对话框指针, 并改写构造函数。

```
class CClientSocket : public CSocket
{
public:
 public:
 CClientSocket(CKinescodeDlg* pDlg);
 virtual ~CClientSocket();
 CKinescodeDlg* m_pDlg;

 public:
 public:
 virtual void OnReceive(int nErrorCode);

protected:
};
```

(5) 在 CClientSocket 类中改写 OnReceive 方法。当套接字有数据接收时, 将调用对话框类中定义的方法接收数据。

```
void CClientSocket::OnReceive(int nErrorCode)
{
 m_pDlg->ReceiveData(this);//接收数据
}
```

(6) 在对话框类中定义如下成员变量。

|                |                |                |
|----------------|----------------|----------------|
| HWND           | m_hVideoWnd;   | //视频捕捉窗口       |
| CWinThread*    | m_pThread;     | //线程对象指针       |
| CEvent         | m_Event;       | //事件对象,用于线程同步  |
| BOOL           | m_IsKindscode; | //是否录像         |
| CClientSocket* | m_pSock;       | //客户端套接字       |
| BOOL           | m_IsSend;      | //是否发送         |
| BOOL           | m_IsReceived;  | //服务器端接收数据是否完成 |

(7) 处理“发送数据”按钮的单击事件, 设置预览模式的回调函数。

```
void CKinescodeDlg::OnStopkinescode()
{
}
```



```
capSetCallbackOnFrame(m_hVideoWnd,CaptureFun);
m_IsSend = TRUE;
SetTimer(1,200,NULL);
}
```

(8) 处理对话框的 WM\_TIMER 消息, 时时发送数据。

```
void CKinescodeDlg::OnTimer(UINT nIDEvent)
{
 if (m_IsSend)
 {
 if (m_IsSending==FALSE)
 if (m_IsSave)
 ((CKinescodeDlg*)AfxGetMainWnd()->SendData();//发送数据
 }
 CDialog::OnTimer(nIDEvent);
}
```

服务器端应用程序设计步骤如下。

(1) 创建一个基于对话框的应用程序。

(2) 向对话框中添加文本编辑控件、按钮控件。

(3) 从 CSocket 类派生一个子类 CClientSocket, 作为客户端套接字。在 CClientSocket 类中定义一个对话框指针, 并改写构造函数和 OnReceive 方法。

```
CClientSocket::CClientSocket(CUuuuDlg* pDlg)
: m_pDlg(pDlg)
{
}

void CClientSocket::OnReceive(int nErrorCode)
{
 if(m_pDlg)
 {
 //调用对话框的ReceiveData方法
 m_pDlg->ReceiveData(this);
 }
 CSocket::OnReceive(nErrorCode);
}
```

(4) 从 CSocket 类派生一个子类 CServerSocket, 作为服务器端套接字。在 CServerSocket 类中定义一个对话框指针, 并改写构造函数和 OnAccept 方法。

```
CServerSocket::CServerSocket(CUuuuDlg* pDlg)
: m_pDlg(pDlg)
{
}

void CServerSocket::OnAccept(int nErrorCode)
{
 m_pDlg->AcceptConnect();//套接字连接
 CSocket::OnAccept(nErrorCode);
}
```

(5) 向对话框类中添加 ReceiveData 方法, 用于接收客户端传来的数据。

```
void CUuuuDlg::ReceiveData(CClientSocket *sock)
{
 HGLOBAL hGlobal = GlobalAlloc(GMEM_MOVEABLE,999999);//创建缓存空间
 void* lpBuf = GlobalLock(hGlobal);//锁定缓存空间

 DWORD size= sock->Receive(lpBuf,999999);//接收数据
 if (m_IsStop)
 {
 m_file.Open("c:\\b.bmp",CFile::modeCreate|CFile::modeWrite);//打开文件
 m_file.WriteHuge(lpBuf,size);//写入数据
 m_IsStop = FALSE;
 m_IsReceived = FALSE;

 GlobalUnlock(hGlobal);//缓存解锁
 GlobalFree(hGlobal);//缓存释放
 return;
 }
 char* temp = (char*)lpBuf;
 if ((temp[size-1]=='\n'))
 {

```



```

m_file.Close();
m_IsStop = TRUE;
m_IsReceived = TRUE;
char temp[1]={'a'};
sock->Send(temp,1);
}
else
{
 m_file.WriteHuge(lpBuf,size);//写入数据

 GlobalUnlock(hGlobal);
 GlobalFree(hGlobal);
}
}

```

(6) 向对话框类中添加 AcceptConnect 方法, 接受客户端的连接。

```

void CUuuuDlg::AcceptConnect()
{
 //接受客户端的连接
 m_pServerSock->Accept(*m_pClientSock);
}

```

### 举一反三

根据本实例, 读者可以:

- 开发视频聊天软件。

## 实例 334 云台控制

这是一个可以提高基础性能的实例

实例位置: 光盘\mingrisoft\11\334

### 实例说明

云台控制是指对单位、学校或一些公用厂所安装的摄像头进行控制, 通过对摄像头的广角控制就可以监控到摄像头可监测范围内的所有信息。效果如图 11.15 所示。

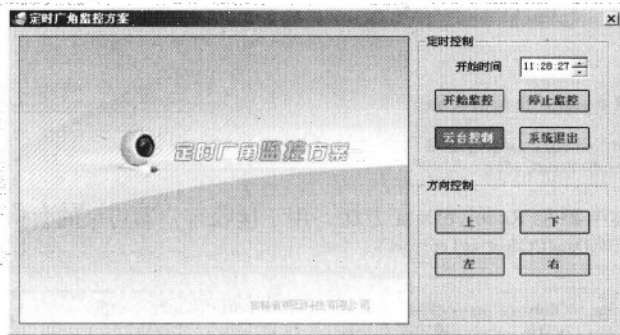


图 11.15 云台控制

### 技术要点

云台是通过云台解码器与计算机串口或并口相连的, 程序通过向云台解码器发送指令来实现云台控制。这里的指令是由云台控制协议确定的。不同的厂家, 云台控制协议也不尽相同。以 Pelco-D2400 为例, 其命令格式由 7 个字节构成。第 1 个字节为同步字节, 始终为 FFH; 第 2 个字节为地址码, 也就是摄像头的逻辑地址号, 范围在 00H~FFH 之间, 是在安装摄像头时手动设置的, 该值一定要正确, 否则命令不会执行; 第 3、4 个字节表示指令码, 即执行哪项操作, 例如, 向上、下、左、右移动摄像头等; 第 5、6 个字节表示数据码, 用于指定摄像头的水平、垂直方向移动速

度；第 7 个字节为校验码，它是由第 2、3、4、5、6 个字节数据之和与 100H 取模获得的。

此外，在执行某一命令后，应执行停止命令，否则命令执行的动作会一直执行。例如，将摄像头向上移动。如果不发送停止命令，摄像头就会一直向上移动。在 Pelco-D2400 协议的停止命令中，第 1 个字节为 FFH；第 2 个字节为地址码；第 3、4、5、6 个字节为 00F；第 7 个字节为第 2 个字节数据与 100H 的模。

只要知道了控制命令，就可以通过向串口发送这些命令来控制云台了。例如，下面的代码实现了云台的向下移动。

```
void CCloudMsgDlg::OnDown()
{
 //向下移动
 unsigned char data[7]= {0xff,0x02,0x00,0x10,0x00,0xff,0x12};
 VARIANT vt;
 SAFEARRAY* pSafe;
 SAFEARRAYBOUND band;
 band.cElements =7;
 band.lLbound = 0;
 pSafe = SafeArrayCreate(VT_UI1,1,&band);
 for (long i = 0; i<7; i++)
 {
 SafeArrayPutElement(pSafe,&i,&data[i]);
 }
 vt.vt= VT_ARRAY|VT_UI1;
 vt.parray = pSafe;
 m_Com.SetOutput((COleVariant)vt);
 //停止移动
 unsigned char stopdata[7]= {0xff,0x02,0x00,0x00,0x00,0x00,0x02};
 for (i = 0; i<7; i++)
 {
 SafeArrayPutElement(pSafe,&i,(void*)&stopdata[i]);
 }
 vt.vt= VT_ARRAY|VT_UI1;
 vt.parray = pSafe;
 m_Com.SetOutput((COleVariant)vt);
}
```

## 实现过程

- (1) 创建一个基于对话框的工程，向对话框中添加标签、按钮、日期等控件。
- (2) 向对话框类中添加如下成员变量：

|                                 |              |
|---------------------------------|--------------|
| //成员变量                          |              |
| unsigned char (*m_pData) [100]; | //端口数据       |
| int m_Len;                      | //云台协议使用的字节数 |
| int m_ActoinCount;              | //云台控制动作数    |
| int m_Port;                     | //端口号        |
| CString m_Setting;              | //端口信息       |
| BOOL m_Tail;                    | //是否开始监控     |
| HANDLE m_hThread;               | //线程句柄       |

- (3) 在对话框初始化时从 INI 文件中读取基础信息。

```
BOOL CTimeDIYDlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
 ASSERT(IDM_ABOUTBOX < 0xF000);

 CMenu* pSysMenu = GetSystemMenu(FALSE);
 if (pSysMenu != NULL)
 {
 CString strAboutMenu;
 strAboutMenu.LoadString(IDS_ABOUTBOX);
 if (!strAboutMenu.IsEmpty())
 {
 pSysMenu->AppendMenu(MF_SEPARATOR);
 pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
 }
 }
}
```

```
SetIcon(m_hIcon, TRUE);
SetIcon(m_hIcon, FALSE);

CTime time = CTime::GetCurrentTime();
m_Time.SetTime(&time);

m_Len = GetPrivateProfileInt("使用字节数", "字节数", 8, ".stage.ini");
m_ActoinCount = GetPrivateProfileInt("控制项", "控制数", 13, ".stage.ini");
m_pData = new unsigned char[m_ActoinCount][100];

m_Port = GetPrivateProfileInt("端口设置", "端口号", 1, ".stage.ini");

GetPrivateProfileString("端口设置", "环境设置",
"9600,n,8,1", m_Setting.GetBuffer(0), MAX_PATH, ".stage.ini");

int data;
char buff[20] = {0};
char var[20] = {0};

for (int i = 0; i < m_ActoinCount; i++)
 for (int j = 0; j < m_Len; j++)
 {
 char section[20] = "字节";
 itoa(i+1, var, 10);
 itoa(j+1, buff, 10);
 strcat(section, buff);
 data = GetPrivateProfileInt(var, section, 0, ".stage.ini");
 m_pData[i][j] = data;
 }

//设置端口信息,并打开端口
m_Com.SetSettings(m_Setting);
m_Com.SetOutBufferSize(512);
m_Com.SetCommPort(m_Port);
m_Com.SetSThreshold(0);
m_Com.SetPortOpen(TRUE);

if (VCAInitSdk(m_hWnd))
{
 VCAREgVidCapCallBack(0, CapCallBack);
 VCAOpenDevice(0, m_Panel.m_hWnd);
 VCAStartVideoPreview(0);
}
m_Tail = FALSE;

SetTimer(1, 300, NULL);
return TRUE;
}
```

(4) 处理对话框的 WM\_WINDOWPOSCHANGED 消息, 在对话框位置改变时更新视频预览窗口。

```
void CTimeDIYDlg::OnWindowPosChanged(WINDOWPOS FAR* lpwndpos)
{
 CDialog::OnWindowPosChanged(lpwndpos);
 VCAUpdateOverlayWnd(m_Panel.m_hWnd);
 VCAUpdateVideoPreview(0, m_Panel.m_hWnd);
}
```

(5) 编写线程函数, 实现系统的自动监控。

```
DWORD WINAPI ThreadProc(LPVOID lpParameter)
{
 CTimeDIYDlg* pDlg = (CTimeDIYDlg*)lpParameter;
 while (true)
 {
 //进行云台控制
 VARIANT vt;
 SAFEARRAY* pSafe;
 SAFEARRAYBOUND band;
 band.cElements = pDlg->m_Len;
 band.lLbound = 0;
 pSafe = SafeArrayCreate(VT_UI1, 1, &band);

 //向上
 for (long i = 0; i < pDlg->m_Len; i++)
 {
 SafeArrayPutElement(pSafe, &i, (void*)&pDlg->m_pData[0][i]);
 }
 }
}
```

```

 }
 vt.vt= VT_ARRAY|VT_UI1;
 vt.parray = pSafe;
 pDlg->m_Com.SetOutput((COleVariant)vt);

 Sleep(15000);

 //向左
 vt.vt= VT_ARRAY|VT_UI1;
 vt.parray = pSafe;
 for (i = 0; i<pDlg->m_Len; i++)
 {
 SafeArrayPutElement(pSafe,&i,(void*)&pDlg->m_pData[2][i]);
 }
 pDlg->m_Com.SetOutput((COleVariant)vt);

 vt.vt= VT_ARRAY|VT_UI1;
 vt.parray = pSafe;
 for (i = 0; i<pDlg->m_Len; i++)
 {
 SafeArrayPutElement(pSafe,&i,(void*)&pDlg->m_pData[1][i]);
 }
 Sleep(15000);
 //向下
 pDlg->m_Com.SetOutput((COleVariant)vt);

 //向右
 Sleep(15000);
 vt.vt= VT_ARRAY|VT_UI1;
 vt.parray = pSafe;
 for (i = 0; i<pDlg->m_Len; i++)
 {
 SafeArrayPutElement(pSafe,&i,(void*)&pDlg->m_pData[3][i]);
 }
 pDlg->m_Com.SetOutput((COleVariant)vt);
 Sleep(15000);
}
return 0;
}

```

(6) 处理“开始监控”按钮的单击事件，让系统自动监控。

```

void CTimeDIYDlg::OnStartTail()
{
 m_Tail = TRUE;
 //开始监控
 VCStartVideoCapture(0,CAP_MPEG4_STREAM,MPEG4_AVIFILE_ONLY,"C:\\\\WW.avi");

 DWORD threadID;
 m_hThread = ::CreateThread(NULL,0,ThreadProc,(LPVOID)this,0,&threadID);
}

```

(7) 处理“停止监控”按钮的单击事件，停止系统自动监控。

```

void CTimeDIYDlg::OnStopTail()
{
 if (m_Tail==TRUE)
 {
 //停止运动
 VARIANT vt;
 SAFEARRAY* pSafe;
 SAFEARRAYBOUND band;
 band.cElements =m_Len;
 band.lLbound = 0;
 pSafe = SafeArrayCreate(VT_UI1,1,&band);

 for (long i = 0; i<m_Len; i++)
 {
 SafeArrayPutElement(pSafe,&i,(void*)&m_pData[12][i]);
 }
 vt.vt= VT_ARRAY|VT_UI1;
 vt.parray = pSafe;
 m_Com.SetOutput((COleVariant)vt);

 ::TerminateThread(m_hThread,0);
 //停止监控
 }
}

```



```

 VCAStopVideoCapture(0);
 m_Tail = FALSE;
 }
}

```

(8) 处理对话框的 WM\_TIMER 消息, 当指定的时间到达时, 实现提供的自动监控。

```

void CTimeDIYDlg::OnTimer(UINT nIDEvent)
{
 CTime time= CTime::GetCurrentTime();

 CTime strtime;
 m_Time.GetTime(strtime);

 if (time==strtime)
 {
 OnStartTail();
 KillTimer(1);
 }
 CDialog::OnTimer(nIDEvent);
}

```

(9) 在对话框关闭时停止摄像头的转动。

```

void CTimeDIYDlg::OnCancel()
{
 delete [] m_pData;
 VCACloseDevice(0);
 VCAUnInitSdk();
 OnStopTail();
}

```

### 举一反三

根据本实例, 读者可以:

- 远程云台控制。

## 11.5 扫描、条形码、POS 控制

随着计算机技术的不断发展, 如今的商场或超市在销售商品时, 已不再采用手工输入的方式, 而是利用条形码扫描器进行商品的入库、销售操作, 这样, 不仅提高了工作效率, 而且提高了工作质量。本节将介绍有关条形码、POS 机的相关知识。

### 实例 335 利用条形码扫描器销售商品

这是一个可以提高基础性能的实例

实例位置: 光盘\mingrisoft\11\335

#### 实例说明

如今的许多超市都利用条形码销售商品。操作员利用扫描器在商品的条形码处一扫, 商品的详细信息就会显示在屏幕中。本例实现了利用条形码销售商品的功能。效果如图 11.16 所示。

#### 技术要点

当利用扫描器扫描条形码时, 条形码数据会显示在当前获得焦点的窗口控件中。例如, 如果当前编辑框获得焦点, 那么条形码数据会显示在编辑框中。然后会向编辑框控件发送回车键按下时的消息。

在程序中只要截获编辑框的 WM\_KEYDOWN 消息, 判断当前按键是否是回车键, 如果是, 读取编辑框中的条形码数据, 并从数据表中根据条形码查询商品信息, 将其显示在列表中。

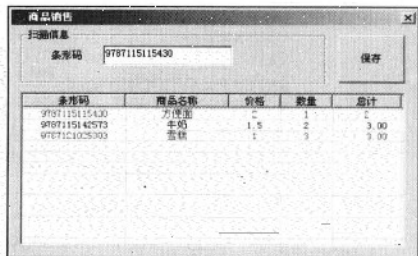


图 11.16 利用条形码扫描器销售商品

```

BOOL CSellGoodsDlg::PreTranslateMessage(MSG* pMsg)
{
 //截获编辑框的WM_KEYDOWN消息
 if ((pMsg->hwnd==m_Barcode.m_hWnd)&&
 (pMsg->message==WM_KEYDOWN))
 if (pMsg->wParam==13)
 OnEditEnter(); //调用自定义的方法
 return CDialog::PreTranslateMessage(pMsg);
}

```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加静态文本控件、按钮控件、文本编辑控件、列表视图控件。
- (3) 在对话框初始化时向列表视图控件中添加列，设置列标题。

```

m_ListInfo.SetTextColor(RGB(255,0,0));
//设置列表扩展风格
m_ListInfo.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_TWOCLICKACTIVATE
|LVS_EX_FULLROWSELECT|LVS_EX_GRIDLINES|LVS_EX_UNDERLINEHOT);
//添加列
m_ListInfo.InsertColumn(1,"条形码",LVCFMT_CENTER,120);
m_ListInfo.InsertColumn(2,"商品名称",LVCFMT_CENTER,120);
m_ListInfo.InsertColumn(3,"价格",LVCFMT_CENTER,60);
m_ListInfo.InsertColumn(4,"数量",LVCFMT_CENTER,60);
m_ListInfo.InsertColumn(5,"总计",LVCFMT_CENTER,80);

```

- (4) 向对话框类中添加 OnEditEnter 方法，读取条形码信息，查询商品信息。

```

void CSellGoodsDlg::OnEditEnter()
{
 CString str;
 m_Barcode.GetWindowText(str);
 if (str.IsEmpty())
 return;

 CString sql;
 sql.Format("select * from MerchandiseInfo where Barcode = '%s'",str);

 m_pRecord->raw_Close();
 m_pRecord->Open((_variant_t)sql,m_pCon.GetInterfacePtr(),
 adOpenDynamic,adLockOptimistic,adCmdText); //打开数据集

 CString c_barcode,c_name,c_price;

 int row = 0;
 CString temp;

 int num;
 float price;

 while (m_pRecord->ADOEOF==FALSE)//循环数据集
 {
 c_barcode =(char*)(_bstr_t) m_pRecord->GetFields()
->GetItem("Barcode")->Value;
 c_name = (char*)(_bstr_t) m_pRecord->GetFields()
->GetItem("MerchandiseName")->Value;
 c_price = (char*)(_bstr_t) m_pRecord->GetFields()->
GetItem("Price")->Value;

 int currow = IsExistInList(c_barcode);
 if (currow==-1) //添加新行
 {
 m_ListInfo.InsertItem(row,"");
 m_ListInfo.SetItemText(row,0,c_barcode);
 m_ListInfo.SetItemText(row,1,c_name);
 m_ListInfo.SetItemText(row,2,c_price);
 m_ListInfo.SetItemText(row,3,"1");
 m_ListInfo.SetItemText(row,4,c_price);

```

```

 }
 else //修改现有行
 {
 temp = m_ListInfo.GetItemText(currow,3); //获取数量
 num = atoi(temp);
 num+=1;
 temp.Format("%d",num);
 m_ListInfo.SetItemText(currow,3,temp);

 temp = m_ListInfo.GetItemText(currow,4); //获取总计
 price = atof(temp)+ atof(c_price);
 temp.Format("%.2f",price);
 m_ListInfo.SetItemText(currow,4,temp);
 }
 m_pRecord->MoveNext();
}
m_Barcode.SetWindowText("");
}

```

(5) 改写对话框的 `PreTranslateMessage` 方法, 截获编辑框的 `WM_KEYDOWN` 消息。

```

BOOL CSellGoodsDlg::PreTranslateMessage(MSG* pMsg)

```

```
{
 //截获编辑框的WM_KEYDOWN消息
 if ((pMsg->hwnd==m_Barcode.m_hWnd)
 &&(pMsg->message==WM_KEYDOWN))
 if (pMsg->wParam==13)
 OnEditEnter();
 return CDialog::PreTranslateMessage(pMsg);
}
```

(6) 处理“保存”按钮的单击事件，将数据保存到数据表中。

```
void CSellGoodsDlg::OnSave()
```

```
{
 if (m_ListInfo.GetItemCount() < 1)
 {
 MessageBox("销售信息不能为空.", "提示", 64);
 return;
 }

 CString sql;
 CString c_barcode, c_num, c_total;
 try
 {
 for (int i = 0; i < m_ListInfo.GetItemCount(); i++)
 {
 c_barcode = m_ListInfo.GetItemText(i, 0);
 c_num = m_ListInfo.GetItemText(i, 3);
 c_total = m_ListInfo.GetItemText(i, 4);
 sql.Format("insert into SellDetail values\n\n('%s', %i, %f)", c_barcode, atoi(c_num), atof(c_total));
 m_pRecord->raw_Close();

 m_pRecord->Open((_variant_t)sql, m_pCon.GetInterfacePtr(),
 adOpenDynamic, adLockOptimistic, adCmdText);
 }
 MessageBox("操作成功");
 m_ListInfo.DeleteAllItems();
 }
 catch (...)
 {
 MessageBox("操作失败");
 }
}
```

### 举一反三

根据本实例，读者可以：

- ### ● 实现超市条形码扫描系统。

## 实例 336

使用数据采集器进行库  
存盘点

本实例是一个非常实用的程序

实例位置: 光盘\mingrisoft\11\336

## 实例说明

现在由于许多的大型仓库存放的商品或原材料都非常多,所以在进行库存盘点时需要很多的时间和人力,为了提高工作效率,可以使用数据采集器检索库存中商品或原材料的数量,然后导入到程序中计算盘点盈亏。程序运行效果如图 11.17 所示。

## 技术要点

采集器中数据的导入首先是利用采集器提供商指定的程序将采集器中的数据导入到本地数据文件中,通常是扩展名为“.dat”的文件。然后分析数据文件的格式,将数据导入到计算机中。在本例中数据文件中存在两列,第一列是条形码的值,第二列是数量。而且每一列的长度是固定的,所以可以定义一个结构对该数据文件进行读取。结构定义如下:

```
typedef struct tagbarcode
{
 char code[25]; //条形码
 char count[14]; //数量
}Barcode;
```

通过这个结构利用 CFile 对象可以一次读取一行的数据,同时提取条形码和数量。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加 1 个列表视图控件;添加 3 个按钮控件,设置其 Caption 属性为“数据导入”、“结算”和“关闭”。
- (3) 在程序初始化时读入盘点表中商品信息及库存数量,实现代码如下:

```
void CBarcodeDlg::InitList()
{
 _RecordsetPtr m_pRecordset;
 m_pRecordset.CreateInstance(__uuidof(Recordset));
 m_pRecordset->Open("select* from 盘点表",
 m_pConnection.GetInterfacePtr(),adOpenKeyset,adLockOptimistic,adCmdText);
 int m = m_pRecordset->GetRecordCount();
 m_list.DeleteAllItems();
 for (int i = 0; i < m; i++)
 {
 m_list.InsertItem(i,(char*)(_bstr_t)m_pRecordset->GetCollect("商品名称"));
 m_list.SetItemText(i,1,(char*)(_bstr_t)m_pRecordset->GetCollect("条形码"));
 m_list.SetItemText(i,2,(char*)(_bstr_t)m_pRecordset->GetCollect("库存数量"));
 m_list.SetItemText(i,3,(char*)(_bstr_t)m_pRecordset->GetCollect("导入库存数量"));
 m_list.SetItemText(i,4,(char*)(_bstr_t)m_pRecordset->GetCollect("盘点盈亏数量"));
 m_pRecordset->MoveNext();
 }
}
```

- (4) 单击“数据导入”按钮将存储在计算机中的采集器数据按条码导入到对应商品的导入库存数量中,实现代码如下:

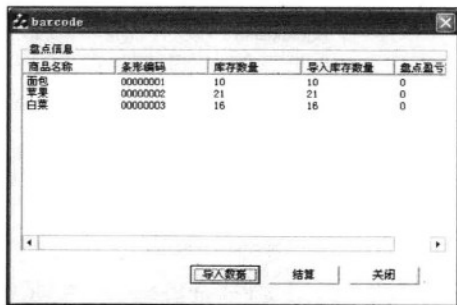


图 11.17 使用数据采集器进行库存盘点



```

void CBarcodeDlg::OnLoadData()
{
 static char BASED_CODE szFilter[] = "Data Files (*.dat)*.dat";

 CFileDialog filedialog(true,NULL,NULL,
 OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
 szFilter,NULL);
 if (filedialog.DoModal() == IDOK)
 {
 CString FileName = filedialog.GetPathName();//获取文件路径
 int Size = sizeof(BarCode);
 BarCode barcode;
 CFile f(FileName, CFile::modeRead);//打开文件

 _CommandPtr m_pCommand;
 m_pCommand.CreateInstance(__uuidof(Command));
 m_pCommand->ActiveConnection = m_pConnection;
 for (int i = 0; i < f.GetLength(); i += Size)
 {
 f.Read(&barcode,Size);//读取数据
 int pos = strcspn(barcode.code, " ");
 barcode.code[pos] = '\0';
 pos = strcspn(barcode.count, " ");
 barcode.count[pos] = '\0';
 CString sql;
 sql.Format("Update 盘点表 set 导入库存数量 = %s,\n
 盘点盈亏数量 = %s - 库存数量 Where 条形码 = %s",
 barcode.count,barcode.count,barcode.code);
 m_pCommand->CommandText = (_bstr_t)sql; //设置SQL语句
 m_pCommand->Execute(NULL, NULL,adCmdText);//执行SQL语句
 }
 f.Close();
 InitList();
 MessageBox("数据导入成功!");
 }
}

```

(5) 单击“结算”按钮将计算库存盘点的盈亏数量，实现代码如下：

```

void CBarcodeDlg::OnBalance()
{
 _CommandPtr m_pCommand;
 m_pCommand.CreateInstance(__uuidof(Command));
 m_pCommand->ActiveConnection = m_pConnection;
 CString sql = "Update 盘点表 Set 库存数量 = 库存数量 + 盘点盈亏数量 \n
 ,盘点盈亏数量 = 0,导入库存数量 = 0";
 m_pCommand->CommandText = (_bstr_t)sql;
 m_pCommand->Execute(NULL, NULL,adCmdText);
 InitList();
 MessageBox("结算成功!");
}

```

### 举一反三

根据本实例，读者可以：

- 实现超市条形码扫描系统。

## 实例 337 设计钱箱控制程序

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\11\337

### 实例说明

为了有效的管理现金的收入和支出，许多超市的收银台利用钱箱存储现金。当客户结账时，

操作员可以通过程序打开钱箱。如何在程序中控制钱箱呢？本例实现了该功能。实例运行结果如图 11.18 所示。

## 技术要点

为了控制钱箱，厂家通常提供相应的 API 函数或一组命令。用户可以通过 API 函数打开钱箱，也可以通过向串口发送特殊的指令打开钱箱，因为 Pos 机通常是通过串口与计算机相连。以中崎微型 Pos 机为例，用户可以通过调用 ZQPntCtrl.dll 动态库中的 OpenMoneyBox 方法打开钱箱，也可以通过向串口发送“Chr\$(27); Chr\$(112); Chr\$(0); Chr\$(5); Chr\$(255)”打开钱箱。本例采用第二种方法。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加静态文本控件、按钮控件、文本编辑控件、列表视图控件、单选按钮控件。
- (3) 向对话框中导入 MSComm ActiveX 控件。
- (4) 在对话框初始化时向列表视图控件中添加列，设置列标题。

```
m_List.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_GRIDLINES|
LVS_EX_FULLROWSELECT|LVS_EX_TWOCLICKACTIVATE);
m_List.InsertColumn(1,"商品名称",LVCFMT_CENTER,100);
m_List.InsertColumn(2,"条形码",LVCFMT_CENTER,100);
m_List.InsertColumn(3,"单价",LVCFMT_CENTER,80);
m_List.InsertColumn(4,"数量",LVCFMT_CENTER,80);
m_List.InsertColumn(5,"金额",LVCFMT_CENTER,80);
```

```
m_List.InsertItem(0,"");
m_List.SetItemText(0,0,"华龙方便面");
m_List.SetItemText(0,1,"4542156854754");
m_List.SetItemText(0,2,"0.8");
m_List.SetItemText(0,3,"10");
m_List.SetItemText(0,4,"8");
```

```
m_List.InsertItem(1,"");
m_List.SetItemText(1,0,"中华牙膏");
m_List.SetItemText(1,1,"4542564854754");
m_List.SetItemText(1,2,"2.4");
m_List.SetItemText(1,3,"10");
m_List.SetItemText(1,4,"24");
```

```
m_List.InsertItem(2,"");
m_List.SetItemText(2,0,"舒肤佳香皂");
m_List.SetItemText(2,1,"4542564854111");
m_List.SetItemText(2,2,"3.5");
m_List.SetItemText(2,3,"20");
m_List.SetItemText(2,4,"70");
```

```
CWnd* pCtrl = GetDlgItem(IDC_COM1);
m_Com.SetCommPort(1);
m_Com.SetPortOpen(TRUE);
((CButton*)pCtrl)->SetCheck(1);
```

- (5) 处理“找零”按钮的单击事件，打开钱箱。

```
void CMoneyBoxDlg::OnOK()
{
 //Chr$(27);Chr$(112);Chr$(0);Chr$(5);Chr$(255) 钱箱控制
 char ctrl[5] = {27,112,0,5,255};
 m_Com.SetOutput((COleVariant)ctrl);
}
```

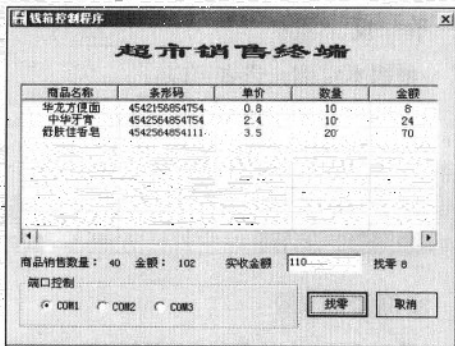


图 11.18 设计钱箱控制程序

## 举一反三

根据本实例，读者可以：

- 实现超市 Pos 机商品销售系统。

## 实例 338 设计扫描仪控制程序

这是一个可以提高基础性能的实例

实例位置：光盘\mingrisoft\11\338

## 实例说明

通过 Photoshop 提供的辅助软件可以控制扫描仪实现图片的扫描，那么如何通过程序控制扫描仪呢？本例实现了该功能，运行程序，单击“开始扫描”按钮，将进行图片扫描。程序运行结果如图 11.19 所示。

## 技术要点

本例主要使用 ActiveX 控件 ImgScan 进行图片扫描。ImgScan 控件的主要方法如下。

- (1) ScannerAvailable 方法。该方法判断扫描仪是否可用。

语法如下：

```
BOOL ScannerAvailable();
```

- (2) OpenScanner 方法。该方法用于打开扫描仪端口。语法如下：

```
long OpenScanner();
```

- (3) StartScan 方法。该方法用于开始扫描。语法如下：

```
long StartScan();
```

- (4) StopScan 方法。该方法用于停止扫描。语法如下：

```
long StopScan();
```

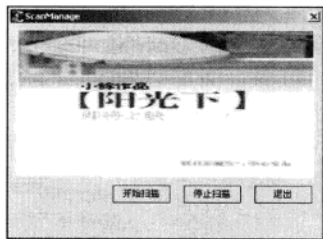


图 11.19 设计扫描仪控制程序

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加按钮控件，并导入 ImgScan、ImgEdit 控件。
- (3) 主要程序代码如下：

```
void CScanManageDlg::OnButton1()
{
 m_Scan.ScannerAvailable();
 m_Scan.OpenScanner();
 m_Scan.StartScan();
}
```

## 举一反三

根据本实例，读者可以：

- 实现批量图纸扫描系统。

## 实例 339 设计发票机控制程序

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\11\339

## 实例说明

许多超市的收银台都是采用发票机打印购物小票。那么在程序中如何控制发票机呢？本例设计了

一个发票机控制程序，单击相应的按钮，可控制发票机打印票据。实例运行结果如图 11.20 所示。

### 技术要点

发票机又称为发票打印机，是一种特殊的点阵打印机，与针式打印机的区别在于，它利用的是小型芯片，分为左右两边，左右两边可以分开打印，也可以同步打印相同的数据。通过设置特殊的指令，可以只打印存根联数据，而不打印收执联数据。

对于发票机的控制，通常包括 4 个动作：打印、跳行、跳页和切纸。用户只要向串口发送特殊的指令，便可以控制打印机的行为。对于每种发票打印机，执行上述动作的指令可能不同，详细说明请参考发票打印机使用说明书。

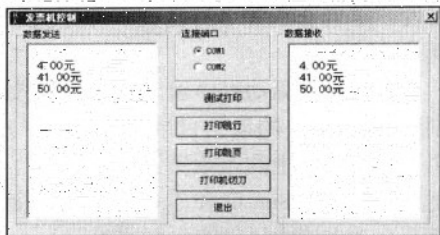


图 11.20 设计发票机控制程序

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加按钮控件、文本编辑控件、单选按钮控件，并导入 MSComm 控件。
- (3) 处理“打印跳行”按钮的单击事件，执行打印跳行。

```
void CPosManageDlg::OnButton2()
{
 m_Com.SetOutput((COleVariant)(CString)(13+10));
}
```

- (4) 处理“打印跳页”按钮的单击事件，实现打印跳页。

```
void CPosManageDlg::OnButton3()
{
 m_Com.SetOutput((COleVariant)(CString)(27+'@'));
 m_Com.SetOutput((COleVariant)(CString)(27+'z'+1));
 m_Com.SetOutput((COleVariant)(CString)(27+'d'+4));
 m_Com.SetOutput((COleVariant)(CString)(""+13+10));
}
```

### 举一反三

根据本实例，读者可以：

- 实现饭店税票打印程序。

## 11.6 语音卡控制

随着语音技术的不断发展，语音卡在通信行业应用非常广泛。本节通过几个典型实例介绍语音卡程序的开发。

### 实例 340 语音卡电话呼叫系统

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\11\340

### 实例说明

随着科学技术的不断发展，语音卡被广泛地应用于商业软件中。本例实现了利用语音卡实现电话呼叫的功能。实例运行结果如图 11.21 所示。

### 技术要点

本例采用东进公司开发的 8 路模拟语音卡，该卡采用灵活的模式化设计，可按需配置外线、



内线两种模块。该卡可实现坐席、会议、FSK 数据收发、语音合成等多种功能,并提供 SDK 开发工具包。

在安装完驱动程序后,相应的动态链接库会复制到 Windows 的系统目录下,程序中正是通过调用这些动态库中的函数操作语音卡的。主要的动态链接库有 NewSig.dll 和 Tc08a32.dll 两个。下面介绍这两个动态库中提供的主要函数。

(1) LoadDRV 函数。该函数用于加载动态链接库。语法如下:

```
long WINAPI LoadDRV(void);
```

返回值: 返回值为 0 表示成功; -1 表示打开设备驱动程序错误; -2 表示在读取 TC08A-V.INI 文件时,发生错误; -3 表示 INI 文件的设置与实际的硬件不一致时,发生错误。

(2) FreeDRV 函数。该函数用于关闭驱动程序。语法如下:

```
void WINAPI FreeDRV(void);
```

(3) EnableCard 函数。该函数用于初始化电话卡的硬件并为每个通道分配语音缓冲区。语法如下:

```
long WINAPI EnableCard(WORD wUsedCh, WORD wFileBufLen);
```

参数说明:

- wUsedCh: 标识通道数量。
- WFileBufLen: 标识分配的缓冲区大小。

(4) CheckValidCh 函数。该函数检测在当前机器内可用的通道总数。语法如下:

```
WORD WINAPI CheckValidCh(void);
```

返回值: 通道数量。

(5) CheckChType 函数。该函数用于测试某个通道的类型。语法如下:

```
WORD WINAPI CheckChType(WORD wChnlNo);
```

参数说明:

- wChnlNo: 标识通道号。
- 返回值: 返回值为 0 表示内线, 为 1 表示外线, 为 2 表示悬空。

(6) PUSH\_PLAY 函数。该函数用于维持文件录音的持续进行, 需在处理函数的大循环中调用。语法如下:

```
void WINAPI PUSH_PLAY (void);
```

(7) SetBusyPara 函数。该函数用于设置要检测的挂机忙音的参数。语法如下:

```
void WINAPI SetBusyPara(WORD BusyLen);
```

参数说明:

- BusyLen: 标识忙音的时间长度, 单位为毫秒。

(8) RingDetect 函数。该函数用于测试外线是否振铃或内线是否提机。语法如下:

```
BOOL WINAPI RingDetect(WORD wChnlNo);
```

参数说明:

- wChnlNo: 标识通道号。
- 返回值: 返回值如果为 1, 对于外线表示有振铃信息, 对于内线, 表示提机; 返回值如果为 0, 对于外线, 表示无振铃信息, 对于内线, 表示挂机。

(9) OffHook 函数。该函数用于外线提机。对于内线, 不起作用。语法如下:

```
void WINAPI OffHook(WORD wChnlNo);
```

参数说明:

- wChnlNo: 标识外线通道。

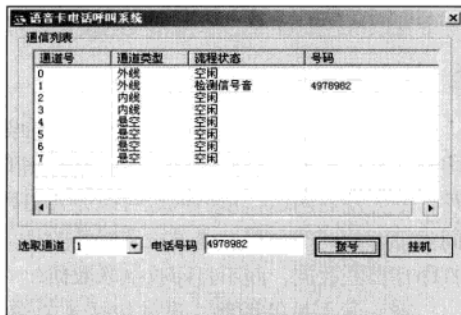


图 11.21 语音卡电话呼叫系统

(10) HangUp 函数。该函数用于外线挂机。对于内线，不起作用。语法如下：

```
void WINAPI HangUp(WORD wChnlNo);
```

参数说明：

- wChnlNo：标识外线通道。

(11) Sig\_Init 函数。该函数用于完成信号音检测的初始化工作。语法如下：

```
int WINAPI Sig_Init(WORD wPara);
```

参数说明：

- wPara：默认值为 0，不起作用。

(12) Sig\_CheckBusy 函数。该函数用于忙音检测。语法如下：

```
int WINAPI Sig_CheckBusy(WORD wChNo);
```

参数说明：

- wChNo：标识通道号。
- 返回值：返回值为 1 表示检测到忙音，返回值为 0，表示没有检测到忙音。

(13) Sig\_ResetCheck 函数。该函数用于清空忙音检测的缓冲区以及内部计数。当检测到对方挂机的忙音后，必须调用本函数。语法如下：

```
void WINAPI Sig_ResetCheck(WORD wChNo);
```

参数说明：

- wChNo：标识通道号。

(14) Sig\_StartDial 函数。该函数用于开始拨号。语法如下：

```
int WINAPI Sig_StartDial(WORD wChNo, char* DialNum, char* PreDialNum, WORD wMode);
```

参数说明：

- wChNo：标识通道号。
- DialNum：标识呼出号码。
- PreDialNum：标识前导号码。
- wMode：呼出检测的模式。

(15) Sig\_CheckDial 函数。该函数用于检测呼出结果。语法如下：

```
int WINAPI Sig_CheckDial(WORD wChNo);
```

参数说明：

- wChNo：标识通道号。
- 返回值：返回值为 S\_NORESULT，表示尚未得出结果；为 S\_NODIALTONE，表示没有拨号音；为 S\_BUSY，表示检测到对方占线的忙音；为 \_CONNECT，表示对方摘机，可以进行通话；为 S\_NOBODY，表示振铃若干次，无人接听；为 S\_NOSIGNAL，表示没有信号音。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加按钮、列表视图、组合框 Combo Box 等控件。
- (3) 在对话框初始化时加载驱动程序，检测通道数量、判断通道类型。

```
//设置列表扩展风格
```

```
m_CardList.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_GRIDLINES|LVS_EX_FULLROWSELECT|LVS_EX_TWOCLICKACTIVATE);
```

```
//添加列
```

```
m_CardList.InsertColumn(1,"通道号",LVCFMT_LEFT,80);
m_CardList.InsertColumn(2,"通道类型",LVCFMT_LEFT,80);
m_CardList.InsertColumn(3,"流程状态",LVCFMT_LEFT,120);
m_CardList.InsertColumn(4,"号码",LVCFMT_LEFT,150);
```

```
//加载驱动程序
long result = LoadDRV(); //0成功,
if (result!=0)
{
 MessageBox("加载驱动程序错误");
}
//检查合法的通道
m_Channels = CheckValidCh();
m_pLines = new LINESTRUCT[m_Channels];
//设置忙音0.35秒
SetBusyPara(350);

//初始化电话卡的硬件,并为每个通道分配语音缓冲区
EnableCard(m_Channels,1024*16);
CString convert;
for (int i = 0; i<m_Channels; i++)
{
 m_pLines[i].State = CH_FREE;
 convert.Format("%i",i);
 m_CardList.InsertItem(i,"");
 m_CardList.SetItemText(i,0,convert);
 WORD type = CheckChType(i);
 m_pLines[i].nType = type;

 switch(type)
 {
 case 0: //内线
 {
 convert = "内线";
 break;
 }
 case 1: //外线
 {
 m_Chanel.AddString(convert);
 convert = "外线";

 break;
 }
 case 2: //悬空
 {
 convert = "悬空";
 break;
 }
 }
 m_CardList.SetItemText(i,1,convert);
}
//初始化信号音函数
Sig_Init(0);
SetTimer(1,600,NULL);
```

(4) 处理对话框的 WM\_TIMER 事件,判断当前各通道的状况。

```
void CSoundCardCallDlg::OnTimer(UINT nIDEvent)
{
 //维持断续振铃及信号音的函数
 PUSH_PLAY();
 FeedSigFunc();

 int result;
 for (int i=0; i<m_Channels;i++)
 {
 if (m_pLines[i].State==CH_FREE)
 {
 m_CardList.SetItemText(i,2,"空闲");
 m_CardList.SetItemText(i,3,"");
 }
 switch(m_pLines[i].State)
 {
 case CH_DIAL:
```

```

{
 m_CardList.SetItemText(i,2,"检测信号音");
 m_CardList.SetItemText(i,3,m_CurNumber);
 //检测某路DTMF发送是否结束,为1,已发送完毕,可以开始信号音检测
 if (CheckSendEnd(i)==1)
 {
 StartSigCheck(i);
 m_pLines[i].State=CH_CHECKSIG;
 }
 break;
}
case CH_CHECKSIG:
{
 int tt = Sig_CheckDial(i);
 if(tt == S_BUSY)
 {
 m_pLines[i].State = CH_BUSY;
 }
 else if(tt == S_CONNECT)
 {
 m_pLines[i].State = CH_CONNECT;
 }
 else if(tt == S_NOSIGNAL)
 {
 m_pLines[i].State= CH_NOSIGNAL;
 }
 else if(tt == S_NOBODY)
 {
 m_pLines[i].State= CH_NOBODY;
 }
 else if(tt == S_NODIALTONE)
 {
 m_pLines[i].State= CH_NODIALTONE;
 }
 break;
}
case CH_NODIALTONE: //没有信号音
case CH_NOSIGNAL: //没有信号
case CH_BUSY: //对方占线(检测到忙音)
{
 OnButton2();
 break;
}
}
}
CDialog::OnTimer(nIDEvent);
}

```

(5) 处理“拨号”按钮的单击事件,开始拨号。

```

void CSoundCardCallDlg::OnButton1()
{
 CString str,number;
 m_Chanel.GetWindowText(str);
 m_Number.GetWindowText(number);
 m_CurNumber = number;
 if (str.IsEmpty()||number.IsEmpty())
 return;
 //外线摘机
 m_CurChanel = atoi(str);
 OffHook(atoi(str));
 Sig_StartDial(atoi(str),number.GetBuffer(0),"",0);
 m_pLines[atoi(str)].State=CH_DIAL;
}

```

(6) 处理“挂机”按钮的单击事件,执行挂机操作。

```

void CSoundCardCallDlg::OnButton2()
{
 HangUp(m_CurChanel);
 Sig_ResetCheck(m_CurChanel);
}

```



```
StartSigCheck(m_CurChanel);
m_pLines[m_CurChanel].State = CH_FREE;
}
```

## 举一反三

根据本实例，读者可以：

- 实现电话自助服务系统。

## 实例 341 语音卡实现来电显示

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\11\341

## 实例说明

随着市场竞争的加剧，企业越来越重视客户服务和市场反馈。本例实现了电话来电显示支持功能。当有客户打入电话时，会读取客户的电话号码，根据电话号码可以提取客户的相关信息，方便客服人员有针对性地进行服务。实例运行结果如图 11.22 所示。

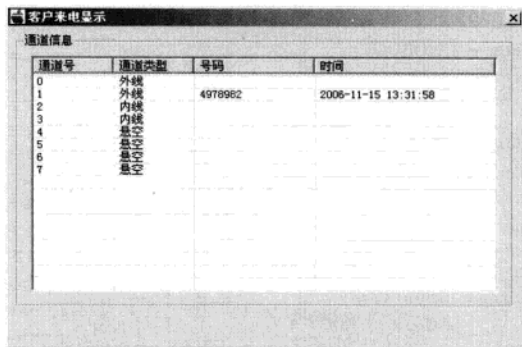


图 11.22 语音卡实现来电显示

## 技术要点

在语音卡开发包的动态库中提供了 GetCallerIDStr 函数用于获取主叫号码。该函数语法如下：

WORD WINAPI GetCallerIDStr (WORD wChnlNo, LPSTR IDStr);

参数说明：

- wChnlNo：标识通道号。
- IDStr：用于接收读取的号码。
- 返回值：返回值为 0，表示未收到任何信息；为 1，表示正在接收头信息；为 2 表示正在接收 ID 号码；为 3 表示接收完毕，校验正确；为 4 表示接收完毕，校验错误。

在调用 GetCallerIDStr 函数时，只有返回值为 3 或 4 才表示已经正确接收了主机号码。

示例如下：

```
void CIncomingShowDlg::ReadNumber(WORD nID)
```

```
{
 char number[100];
 int result;
 PUSH_PLAY();
 FeedSigFunc();

 memset(number,0,100);
 if ((RingDetect(nID)==FALSE))
```

```

{
 m_pLines[nID].m_BusySum+=1;
}
else
 m_pLines[nID].m_BusySum = 0;

result = GetCallerIDStr(nID,number);//获取来电号码
GetDtmfCode(nID);

if ((result==3)||(result==4))
{
 m_CurNumber = number;
 m_CurNumber = m_CurNumber.Mid(8);
 m_pLines[nID].CallerID =m_CurNumber;
 m_CardList.SetItemText(nID,2,m_CurNumber);
 if (m_CurTime.IsEmpty())
 m_CurTime= CTime::GetCurrentTime().
Format("%Y-%m-%d %H:%M:%S");
 m_CardList.SetItemText(nID,3,m_CurTime);
 InitDtmfBuf(nID);
}
}

```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加列表视图控件。
- (3) 在对话框初始化时加载驱动程序，检测通道数量、判断通道类型。

```

//加载驱动程序
if (LoadDRV()!=0)
 MessageBox("加载驱动程序错误");

//检查可用的通道数量
m_ChanelCount = CheckValidCh();

SetBusyPara(350);

EnableCard(m_ChanelCount,1024*59);

//设置列表扩展风格
m_CardList.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_GRIDLINES|
LVS_EX_FULLROWSELECT|LVS_EX_TWOCLICKACTIVATE);

//添加列
m_CardList.InsertColumn(1,"通道号",LVCFMT_LEFT,80);
m_CardList.InsertColumn(2,"通道类型",LVCFMT_LEFT,80);
m_CardList.InsertColumn(3,"号码",LVCFMT_LEFT,120);
m_CardList.InsertColumn(4,"时间",LVCFMT_LEFT,180);

m_pLines = new LINESTRUCT[m_ChanelCount];

CString convert;
for (int i = 0; i<m_ChanelCount; i++)
{
 m_pLines[i].State = CH_FREE;
 m_pLines[i].IsReceiving = FALSE;
 convert.Format("%i",i);
 m_CardList.InsertItem(i,"");
 m_CardList.SetItemText(i,0,convert);
 WORD type = CheckChType(i);
 m_pLines[i].nType = type;
 m_pLines[i].m_BusySum = 0;
 m_pLines[i].Initied = FALSE;

 switch(type)

```

```

 {
 case 0: //内线
 {
 convert = "内线";
 break;
 }
 case 1: //外线
 {
 //m_Chanel.AddString(convert)
 convert = "外线";

 break;
 }
 case 2: //悬空
 {
 convert = "悬空";
 break;
 }
 }

 m_CardList.SetItemText(i,l,convert);
 }
 //初始化信号音函数
 Sig_Init(0);
 SetDialPara(1000,2000,350,7);

```

SetTimer(1,1,NULL);

#### (4) 向对话框类中添加 ReadNumber 方法, 读取号码。

```

void CIncomingShowDlg::ReadNumber(WORD nID)
{
 char number[100];
 int result;
 PUSH_PLAY();
 FeedSigFunc();

 memset(number,0,100);
 if ((RingDetect(nID)==FALSE))
 {
 m_pLines[nID].m_BusySum+=1;
 }
 else
 m_pLines[nID].m_BusySum = 0;

 result = GetCallerIDStr(nID,number);//获取来电
 GetDtmfCode(nID);

 if ((result==3)||(result==4))//获取来电成功
 {
 m_CurNumber = number;
 m_CurNumber = m_CurNumber.Mid(8);
 m_pLines[nID].CallerID = m_CurNumber;
 m_CardList.SetItemText(nID,2,m_CurNumber);
 if (m_CurTime.IsEmpty())
 m_CurTime= CTime::GetCurrentTime().Format(
"%Y-%m-%d %H:%M:%S");
 m_CardList.SetItemText(nID,3,m_CurTime);
 InitDtmfBuf(nID);
 }
}

```

#### (5) 处理对话框的 WM\_TIMER 消息, 时时检测通道是否有电话打入。

```

void CIncomingShowDlg::OnTimer(UINT nIDEvent)
{
 PUSH_PLAY();
 FeedSigFunc();

 CString str;
 int result;
 BOOL bOffHook = FALSE;
 short code;
 for (int i=0; i<m_ChanelCount;i++)
 {
 if (m_pLines[i].IsReceiving)
 {
 StartSigCheck(i);//信号审核
 if (!m_pLines[i].CallerID.IsEmpty())
 {

```

```

 if (m_pLines[i].m_BusySum>300)
 {
 Sig_ResetCheck(i);
 StartSigCheck(i);
 m_pLines[i].IsReceiving = FALSE;
 m_pLines[i].CallerID = "";
 m_CardList.SetItemText(i,2,"");
 m_CardList.SetItemText(i,3,"");
 m_CurTime = "";
 m_pLines[i].m_BusySum = 0;
 return;
 }
 }
 ReadNumber(i);//获取来电
}
if (RingDetect(i))//有来电
{
 m_pLines[i].m_BusySum = 0;
 if (m_pLines[i].IsReceiving==FALSE)
 {
 StartSigCheck(i);
 ResetCallerIDBuffer(i);//清空来电缓冲区
 StartTimer(i,4);
 ReadNumber(i);//读取来电

 m_pLines[i].IsReceiving = TRUE;
 }
}
}
CDialog::OnTimer(nIDEvent);
}

```

### 举一反三

根据本实例，读者可以：

- 实现客户反馈电话录音系统。

## 实例 342 利用语音卡实现电话录音

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\11\342

### 实例说明

如今的许多电话都具有电话录音的功能。当有电话打入时，如果长时间无人接听，电话会提示对方“主人不在，请您留言”之类的信息。本例实现了该功能，当有电话打入时，如果一段时间后无人接听，将提示对方留言。实例运行结果如图 11.23 所示。

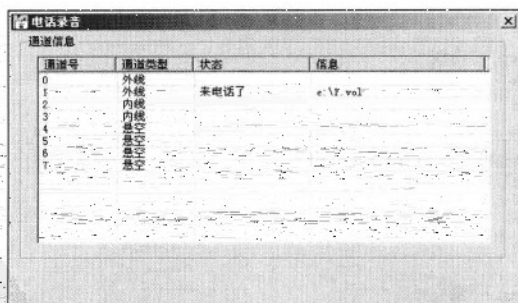


图 11.23 利用语音卡实现电话录音

### 技术要点

在语音卡开发包的动态库中提供了 StartRecordFile 函数用于获取主叫号码。语法如下：



BOOL WINAPI StartRecordFile ( WORD wChnNo, LPSTR FileName, DWORD dwRecordLen );

参数说明:

- wChnNo: 标识录音的通道号。
- FileName: 标识录音的文件名。
- dwRecordLen: 标识文件大小。

在语音卡开发包的动态库中也提供了 StopRecordFile 函数用于停止录音。该函数语法如下:

void WINAPI StopRecordFile (WORD wChnNo);

参数说明:

- wChnNo: 标识要停止的录音通道。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加列表视图控件。
- (3) 在对话框初始化时加载驱动程序, 检测通道数量、判断通道类型。

```
//加载驱动程序
if (LoadDRV()!=0)
 MessageBox("加载驱动程序错误");

//检查可用的通道数量
m_ChannelCount = CheckValidCh();

SetBusyPara(350);

EnableCard(m_ChannelCount, 1024*59);

//设置列表扩展风格
m_CardList.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_GRIDLINES|
LVS_EX_FULLROWSELECT|LVS_EX_TWOCLICKACTIVATE);

//添加列
m_CardList.InsertColumn(1, "通道号", LVCFMT_LEFT, 80);
m_CardList.InsertColumn(2, "通道类型", LVCFMT_LEFT, 80);
m_CardList.InsertColumn(3, "状态", LVCFMT_LEFT, 120);
m_CardList.InsertColumn(4, "信息", LVCFMT_LEFT, 180);

CString convert;
for (int i = 0; i < m_ChannelCount; i++)
{
 convert.Format("%i", i);
 m_CardList.InsertItem(i, "");
 m_CardList.SetItemText(i, 0, convert)
 WORD type = CheckChType(i);

 switch(type)
 {
 case 0: //内线
 {
 convert = "内线";
 break;
 }
 case 1: //外线
 {
 //m_Channel.AddString(convert)
 convert = "外线";

 break;
 }
 case 2: //悬空
 {
 convert = "悬空";
 break;
 }
 }
 m_CardList.SetItemText(i, 1, convert);
}

SetPackRate(PACK_32KBPS);
```

//初始化信号音函数

Sig\_Init(0);

(4) 处理对话框的 WM\_TIMER 消息, 在检测到通道中有电话打入时, 进行录音。

void CPhoneRecordDlg::OnTimer(UINT nIDEvent)

```
{
 PUSH_PLAY();
 FeedSigFunc();
 CString sfile;
 switch(nIDEvent)
 {
 case 1:
 for (int i = 0; i < m_ChanelCount; i++)
 {
 if (RingDetect(i) && CheckChType(i) == 1) //外线
 {
 m_CardList.SetItemText(i, 2, "来电话了");
 //摘机, 开始录音
 OffHook(i);
 StartSigCheck(i);
 //对方没有挂机
 if (ReadCheckResult(i, RECORD_CHECK) != R_BUSY)
 {
 if (m_IsOffHook == TRUE)
 {
 m_IsOffHook = FALSE;
 sfile.Format("c:\\%i.vol", i);
 m_CardList.SetItemText(i, 3, sfile);
 StartRecordFile(i, sfile.GetBuffer(0), 600*1024); //开始录音
 }
 }
 }
 else
 {
 StopRecordFile(i); //停止录音
 m_IsOffHook = TRUE;
 Sig_ResetCheck(i);
 }
 if (CheckRecordEnd(i))
 {
 StopRecordFile(i); //停止录音
 m_IsOffHook = TRUE;
 }
 }
 break;
 }
 CDialog::OnTimer(nIDEvent);
}
```

### 举一反三

根据本实例, 读者可以:

- 利用语音卡实现产品报价。

## 实例 343 利用语音卡实现点歌祝福

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingr\soft\11\343

### 实例说明

随着计算机技术和通信技术的发展, 如今的电话点歌已不再是新鲜事儿了。当用户打入点歌台的电话后, 输入相应的数字, 就会完成点歌任务。本例模拟了电话点歌业务, 当有电话打入后, 输入“136”, 将会听到系统播放的音乐。效果如图 11.24 所示。

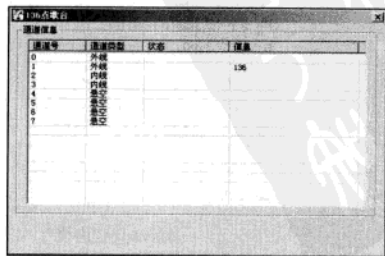


图 11.24 利用语音卡实现点歌祝福

## 技术要点

本例中需要解决：用户在打入电话后如何读取用户的按键信息？如何播放音乐？

在语音卡开发包的动态库中提供了 GetDtmfCode 函数用于获得用户的按键数据。该函数语法如下：

```
short WINAPI GetDtmfCode(WORD wChnlNo);
```

参数说明：

- wChnlNo：标识通道号。
- 返回值：返回值为-1 表示没有按键信息，为 1~9 表示数字 1~9，为 10 表示 0，为 11 表示\*，为 12 表示#，为 13 表示 A，为 14 表示 B，为 15 表示 C，为 0 表示 D。

为了播放音乐，开发包提供了 StartPlayFile 函数。该函数语法如下：

```
BOOL WINAPI StartPlayFile (WORD wChnlNo, LPSTR FileName, DWORD StartPos);
```

参数说明：

- wChnlNo：表示通道号。
- FileName：表示语音文件名称。
- StartPos：表示放音的起始位置。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加列表视图控件。
- (3) 在对话框初始化时加载驱动程序，检测通道数量、判断通道类型。

```
//加载驱动程序
if (LoadDRV()!=0)
 MessageBox("加载驱动程序错误");

//检查可用的通道数量
m_ChannelCount = CheckValidCh();

m_pChanel = new LINESTRUCT[m_ChannelCount];
SetBusyPara(350);

EnableCard(m_ChannelCount,1024*59);

//设置列表扩展风格
m_CardList.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_GRIDLINES|LVS_EX_FULLROWSELECT|LVS_EX_
TWOCLICKACTIVATE);

//添加列
m_CardList.InsertColumn(1,"通道号",LVCFMT_LEFT,80);
m_CardList.InsertColumn(2,"通道类型",LVCFMT_LEFT,80);
m_CardList.InsertColumn(3,"状态",LVCFMT_LEFT,120);
m_CardList.InsertColumn(4,"信息",LVCFMT_LEFT,180);

CString convert;
for (int i = 0; i<m_ChannelCount; i++)
{
 convert.Format("%i",i);
 m_CardList.InsertItem(i,"");
 m_CardList.SetItemText(i,0,convert);
 WORD type = CheckChType(i);//获取通道类型
 m_pChanel[i].RING = FALSE;
 m_pChanel[i].IsSong = FALSE;
 m_pChanel[i].Inited = FALSE;
 switch(type)
 {
 case 0: //内线
 {
 convert = "内线";
 break;
 }
 case 1: //外线
```

```

 {
 //m_Chanel.AddString(convert)
 convert = "外线";

 break;
 }
 case 2: //悬空
 {
 convert = "悬空";
 break;
 }
 }
 m_CardList.SetItemText(i,1,convert);
}
//初始化信号音函数
Sig_Init(0);
SetDialPara(1000,2000,350,7);

SetTimer(1,1,NULL);

```

(4) 处理对话框的 WM\_TIMER 消息，在检测到通道中有电话打入时，读取用户的按键，如果为“136”，将播放语音文件。

```

void CSongForLuckDlg::OnTimer(UINT nIDEvent)
{
 PUSH_PLAY();
 FeedSigFunc();
 int result ;
 short code;
 for (int i=0; i< m_ChanelCount; i++)
 {
 StartSigCheck(i);

 //外线响铃
 if (RingDetect(i)&& CheckChType(i)==1)
 {
 m_pChanel[i].RING = TRUE;
 }

 if (m_pChanel[i].RING == TRUE)
 {
 if (m_pChanel[i].Initied==FALSE)
 {
 OffHook(i);//挂机
 InitDtmfBuf(i);

 m_pChanel[i].Initied = TRUE;
 code = GetDtmfCode(i);
 return;
 }

 while((code = GetDtmfCode(i))!= -1)
 {
 m_PressCh.Insert(m_PressCh.GetLength(),ConvertToASCII(code));
 m_CardList.SetItemText(i,3,m_PressCh);
 }

 if (m_pChanel[i].IsSong ==FALSE&& m_PressCh=="136")
 {
 m_pChanel[i].IsSong = TRUE;
 StartPlayFile(i,"Welcomeold",0L);//播放文件
 }

 if (m_IsSong ==TRUE && CheckPlayEnd(i))
 {
 m_PressCh = "";
 m_pChanel[i].IsSong = FALSE;
 m_pChanel[i].RING = FALSE;
 m_pChanel[i].Initied = FALSE;
 m_CardList.SetItemText(i,3,m_PressCh);
 StopPlayFile(i);//停止文件播放
 }
 }
 }
}

```



```
}
CDialog::OnTimer(nIDEvent);
}
```

### 举一反三

根据本实例，读者可以：

- 利用语音卡实现产品介绍。

## 11.7 手机程序开发

如今手机已成为大众的交流工具。有关手机的程序开发越来越广泛，本节通过几个典型实例介绍如何利用短信猫发送和接收短信。

### 实例 344 利用短信猫发送短信

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\11\344

#### 实例说明

短信猫是利用 SIM 卡发送短信的硬件设备，通过串口或 USB 接口与计算机相连。在程序中可以利用短信猫发送短信。本例实现了利用短信猫发送短信的功能。实例运行结果如图 11.25 所示。

#### 技术要点

本例使用的是人大金仓的串口短信猫。在购买短信猫时会附带有 SDK 的开发包，其中提供了操作短信猫的函数（封装在 dllforvc.dll 动态库中）。下面介绍操作短信猫的主要函数。

(1) GSMModemInitNew 函数。该函数用于初始化短信猫。语法如下：

```
BOOL DLLFORVC_API __stdcall GSMModemInitNew(char *device, char *baudrate,
char *initstring, char *charset, bool swHandshake, char *sn)
```

参数说明：

- device：标识通信端口，如果为 NULL，系统会自动检测。
- baudrate：标识通信波特率，如果为 NULL，系统自动检测。
- initstring：标识初始化命令，为 NULL 即可。
- charset：标识通信字符集，为 NULL 即可。
- swHandshake：标识是否进行软件握手，为 FALSE 即可。
- sn：标识短信猫的授权号，需要根据实际情况填写。

(2) GSMModemSMSsend 函数。该函数用于发送手机短信。语法如下：

```
BOOL DLLFORVC_API __stdcall GSMModemSMSsend(char *serviceCenterAddress,
Int encodeval, char *text, int textlen, char *phonenumber, bool requestStatusReport);
```

参数说明：

- serviceCenterAddress：标识短信中心号码，为 NULL 即可。
- encodeval：标识短信信息编码格式，如果为 8，表示中文短信编码。
- text：标识短信内容。

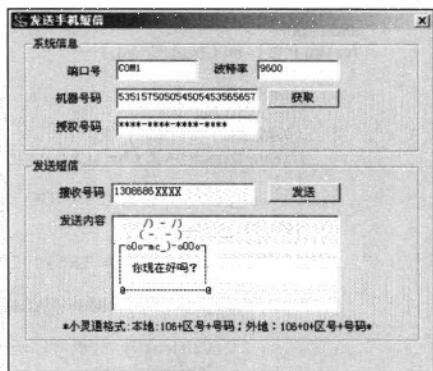


图 11.25 利用语音卡实现点歌祝福

- textlen: 标识短信内容的长度。
- phonenumber: 标识接收短信的电话号码。
- requestStatusReport: 标识状态报告。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加静态文本控件、文本编辑控件、按钮控件。
- (3) 处理“获取”按钮的单击事件，获取短信猫信息。

```
void CSendMsgDlg::OnGetinfo()
{
 m_MerID.Format("%s", GSMModemGetSnInfoNew(NULL, NULL)); //SN信息
 m_Baud.Format("%s", GSMModemGetBaudrate());
 m_Port.Format("%s", GSMModemGetDevice()); //设备名称
 UpdateData(FALSE);
}
```

- (4) 处理“发送”按钮的单击事件，开始发送短信。

```
void CSendMsgDlg::OnSendinfo()
{
 UpdateData(TRUE);
 //初始化猫
 if (GSMModemInitNew((char*)(LPCTSTR)m_Port, (char*)(LPCTSTR)m_Baud,
 NULL, NULL, FALSE, (char*)(LPCTSTR)m_Accredit) == FALSE)
 {
 MessageBox((char*)GSMModemGetErrorMsg()); //错误信息
 return;
 }
 //发送信息
 if (GSMModemSMSsend(NULL, 8, (char*)(LPCTSTR)m_Content,
 m_Content.GetLength(), (char*)(LPCTSTR)m_ReceiverID, FALSE) == FALSE)
 {
 MessageBox("发送失败");
 }
 else
 {
 MessageBox("发送成功");
 }
}
```

## 举一反三

根据本实例，读者可以：

- 利用短信猫群发短信。

## 实例 345 利用短信远程关闭计算机

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\11\345

## 实例说明

本例实现了利用短信远程关闭计算机的功能。运行程序，向短信猫发送“关机”数据，系统将会自动关机。效果如图 11.26 所示。

## 技术要点

实现本例非常容易，只要能够获得短信猫接收的内容就可以了。在短信猫的开发包中提供了 GSMModemSMSReadAll 函数用于获得短信猫中的所有信息。该函数语法如下：

```
BSTR DLLFORVC_API __stdcall GSMModemSMSReadAll(int RD_opt);
```

参数说明：

- RD\_opt: 标识读取短信后是否删除。为 0 将删除短信, 为 1 只读取短信。
- 返回值: 函数返回所有的短信内容, 其格式为“手机号码|短信内容||手机号码|短信内容||……”。

为了能够读取短信猫当前接收的短信, 可以在获取短信内容时, 将短信删除, 这样, 短信猫中就没有已存短信了, 在调用 GSMModemSMSReadAll 函数时, 始终获得的都是当前短信猫接收的短信。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加静态文本控件、文本编辑控件、按钮控件、IP 地址控件。
- (3) 处理对话框的 WM\_TIMER 消息, 判断短信猫中是否有短信需要读取。

```
void CRemoteCloseDlg::OnButton2()
{
 char* reData;
 //获取接收的所有数据
 reData = (char*)GSMModemSMSReadAll(0);
 if (reData != NULL)
 {
 //获取所有短信内容的总长度
 int allDataLen = strlen(reData);

 int num = 0, content = 0;

 char number[30], msgContent[512]; //电话号码, 短信内容
 memset(number, 0, 30);
 memset(msgContent, 0, 512);

 int once = 0; //接收完一个电话号码和一条短信
 for (int i = 0; i < allDataLen; i++)
 {
 if (reData[i] == '|')
 {
 once++;
 if (once == 3 && reData[i-1] == '|')
 {
 CString str = msgContent;

 if (str == "关机")
 {
 KillTimer(1);
 m_Phone = number;
 OnOK();
 if (m_IsSucc) //已成功关机
 {
 GSMModemSMSsend(NULL, 8, "已经成功关机",
 12, (char*)(LPCTSTR)m_Phone, FALSE);
 return;
 }
 else
 {
 GSMModemSMSsend(NULL, 8, "关机失败, 请重新关机",
 20, (char*)(LPCTSTR)m_Phone, FALSE);
 SetTimer(1, 500, NULL);
 }
 }
 memset(number, 0, 30);
 memset(msgContent, 0, 512);
 num = 0;
 content = 0;
 once = 0;
 continue;
 }
 }
 if (once == 0)
 number[num++] = reData[i];
 else if (once == 1)
 msgContent[content++] = reData[i];
 }
 }
}
```

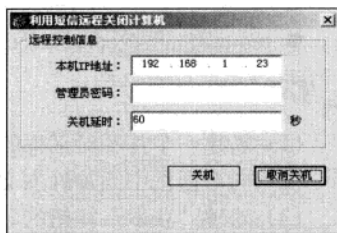


图 11.26 利用短信远程关闭计算机

```

 if (reData[i]!='\r')
 msgContent[content++] = reData[i];
 }
}

void CRemoteCloseDlg::OnTimer(UINT nIDEvent)
{
 OnButton2();
 CDialog::OnTimer(nIDEvent);
}

```

### 举一反三

根据本实例，读者可以：

- 利用短信实现客户资料查询。

## 实例 346 使用“猫”拨打电话

本实例是一个非常实用的程序

实例位置：光盘\mingrisoft\11\346

### 实例说明

在许多客服系统中都使用了客户电话录音、电话回播等功能，这些功能可以通过通信设备猫来完成。本实例实现了使用猫拨打电话的功能，效果如图 11.27 所示。

### 技术要点

在使用 TAPI 函数时，首先需要引用“tapi.h”头文件，然后连接“tapi32.lib”库文件。实现拨号功能主要通过调用 lineMakeCall 函数实现，在使用该函数前，需要对线路进行初始化，并打开线路。这可以通过 lineInitialize 函数和 lineOpen 函数实现。

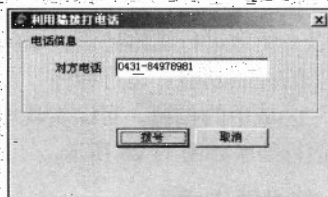


图 11.27 使用猫拨打电话

### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加一个静态文本控件，设置 Caption 属性为“对方电话”；添加一个文本编辑框控件；添加两个按钮控件，设置 Caption 属性为“拨号”和“取消”。

(3) 主要程序代码如下：

```

//对线路进行初始化，并打开线路
LINEINITIALIZEEXPARAMS lineInfo;
lineInfo.dwTotalSize = sizeof(LINEINITIALIZEEXPARAMS);
lineInfo.dwNeededSize = sizeof(LINEINITIALIZEEXPARAMS);
lineInfo.dwUsedSize = sizeof(LINEINITIALIZEEXPARAMS);
CString AppName = "Telephone";
LONG result = lineInitialize(&m_hApp, NULL, lineCallbackFunc,
AppName.GetBuffer(0), &m_tNum);
if (result < 0)
{
 MessageBox("初始化失败");
 LINEDEVCAPS lineCaps;
 lineCaps.dwTotalSize = sizeof(lineCaps);
 lineCaps.dwNeededSize = sizeof(lineCaps);
 lineCaps.dwUsedSize = sizeof(lineCaps);
 lineCaps.dwDevSpecificSize = sizeof(lineCaps);
 m_Reversion = 0;
 LINECALLPARAMS lineParams;
 DWORD temp;
 result = lineNegotiateAPIVersion(m_hApp, 0, 0, 0, &m_Reversion, NULL);
 result = lineOpen(m_hApp, 0, &m_hLine, 0, 0,

```



```

LINECALLPRIVILEGE_OWNER|LINECALLPRIVILEGE_MONITOR,LINECALLMODE_DATAMODEM,NULL);
//实现拨号
HCALL hCall;
LINECALLPARAMS callParams;
CString str;
m_Number.GetWindowText(str);
if (lineMakeCall(m_hLine,&hCall,str.GetBuffer(0),0,&callParams)<0)
 MessageBox("呼叫错误");

```

## 举一反三

根据本实例，读者可以：

- 使用猫进行电话录音。

## 11.8 其他程序

### 实例 347

### 利用神龙卡制作练歌房程序

这是一个可以提高基础性能的实例

实例位置：光盘\mingrisoft\11\347

## 实例说明

在开发酒店、宾馆的点歌系统时，使用神龙卡可以方便地进行媒体控制。本例利用神龙卡实现了音乐的控制功能。运行程序，单击窗口中“控制列表”区域中的按钮，可以实现相应的控制功能。实例运行结果如图 11.28 所示。

## 技术要点

本例使用神龙卡提供的 IREALmagicCtrl 控件实现对媒体的控制。该控件主要方法如下：

(1) SetFilename 方法。该方法用于设置播放的文件名称。语法如下：

```
void SetFilename(LPCTSTR lpszNewValue);
```

参数说明：

- lpszNewValue：标识文件名称。

(2) Play 方法。该方法用于播放指定的媒体文件。语法如下：

```
void Play();
```

(3) GetCurrentFrame 方法。该方法用于获取当前帧的位置。语法如下：

```
long GetCurrentFrame();
```

(4) SetCurrentFrame 方法。该方法用于设置当前帧的位置。语法如下：

```
void SetCurrentFrame(long nNewValue);
```

参数说明：

- nNewValue：标识帧的位置。

(5) SetAudioChannel 方法。该方法用于设置声道。语法如下：

```
void SetAudioChannel(long nNewValue);
```

参数说明：

- nNewValue：标识设置的声道，为 0 表示左声道，为 1 表示右声道，为 2 表示立体声。

(6) Pause 方法。该方法用于暂停媒体文件的播放。语法如下：

```
void Pause();
```

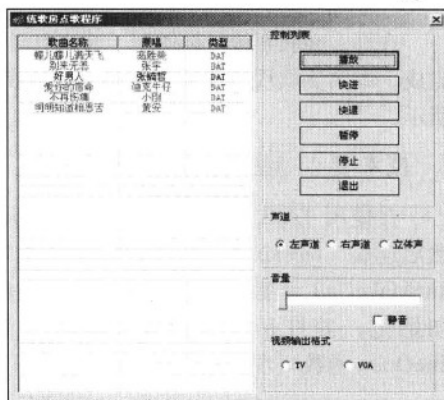


图 11.28 利用神龙卡制作练歌房程序

(7) Stop 方法。该方法用于停止媒体文件的播放。语法如下：

```
void Stop();
```

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 向对话框中添加静态文本控件、按钮控件、单选按钮控件、滑块控件、列表视图控件。
- (3) 通过神龙卡提供的 OCX 文件导入 IREALmagicCtrl 控件。
- (4) 处理“播放”按钮的单击事件，开始播放音乐。

```
//播放
void CSongManageDlg::OnPlay()
{
 CString str;
 int currow = m_SongList.GetSelectionMark();
 if (currow != -1)
 {
 m_Play.Stop();//停止播入
 str = m_SongList.GetItemText(currow,0);
 str+=".dat";
 m_Play.SetFilename(str);//设置播放文件
 m_Play.Play();//播入文件
 }
}
```

- (5) 处理“快进”按钮的单击事件，执行快进操作。

```
//快进
void CSongManageDlg::OnSpeed()
{
 m_Play.SetCurrentFrame(m_Play.GetCurrentFrame()+125); //设置当前帧
}
```

## 举一反三

根据本实例，读者可以：

- 实现练歌房卡拉 OK 系统。

## 实例 348 指纹识别

这是一个可以提高基础性能的实例

实例位置：光盘\mingrisoft\11\348

## 实例说明

指纹头用来采集指纹图片并将其转换成字符串存入系统中，当需要进行识别比对时就可利用当前指纹字符串与系统中已存在的指纹字符串实行对比。本实例所使用的指纹头设备是 uru4000，利用一个 ActiveX 组件进行开发。实例运行结果如图 11.29 所示。

## 技术要点

对指纹头设备的操作是通过 ActiveX 开发组件 Biokey.ocx 实现的。在该组件中定义了操作指纹头的类 CZKFPEngX，在该类中对于指纹头的操作主要通过处理 OnEnroll、OnImage Received、OnCapture、OnFeatureInfo 几个事件来完成。

OnEnroll 事件表示用户登记指纹结束时调用该事件，ActionResult = TRUE 表示成功登记，

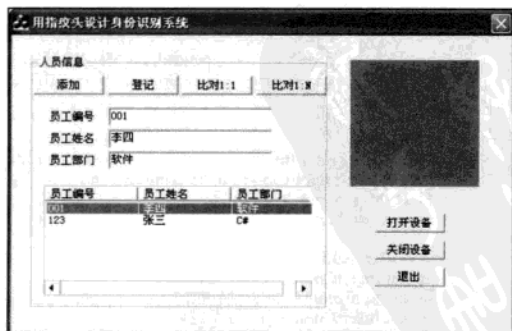


图 11.29 指纹识别

用 pTemplate 属性可取得指纹特征模版; FALSE 表示失败。

OnImageReceived 事件表示设备取到指纹图像或者通过 AddImageFile 和 AddBitmap 加入指纹图像时调用该事件, AImageValid 表示是否进行模板提取, 设置为 FALSE 后, 系统在取到指纹图像后返回, 不进行模板提取。

OnCapture 事件表示 AutoIdentify = FALSE 时, 取到用于比对的指纹验证模板 ATemplate, ActionResult = TRUE 表示成功取到指纹模板; FALSE 表示失败。AutoIdentify = TRUE 时, 返回指纹比对结果(一维数组), 请参考下面定义。

- ATemplate[0]: 代表 ID 值, -1 代表查找失败;
- ATemplate[1]: 返回 1:N 比对分数, 即原来 Identification 函数中的 Score 参数;
- ATemplate[2]: 1:N 指纹比对数;
- ATemplate[3]: 1:1 指纹比对数。

OnFeatureInfo 事件表示取得指纹初始特征, Quality 表示该指纹特征的质量, 有如下可能值:

- 0: 好的指纹特征;
- 1: 特征点不够;
- 2: 其他原因导致不能取到指纹特征。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在窗体上添加 7 个按钮控件, 设置 Caption 属性为“添加”、“登记”、“对比 1:1”、“对比 1:N”、“打开设备”、“关闭设备”和“退出”; 添加 3 个静态文本控件, 设置 Caption 属性为“员工编号”、“员工姓名”和“员工部门”; 添加 3 个文本编辑控件; 添加 1 个列表视图控件和 1 个图片显示控件。

(3) 在窗体初始化时对列表控件进行设置, 插入可用人员列表。代码如下:

```

BOOL CFingerIdentifyDlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 // Add "About..." menu item to system menu

 // IDM_ABOUTBOX must be in the system command range
 ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
 ASSERT(IDM_ABOUTBOX < 0xF000);

 CMenu* pSysMenu = GetSystemMenu(FALSE);
 if (pSysMenu != NULL)
 {
 CString strAboutMenu;
 strAboutMenu.LoadString(IDS_ABOUTBOX);
 if (!strAboutMenu.IsEmpty())
 {
 pSysMenu->AppendMenu(MF_SEPARATOR);
 pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
 }
 }

 // Set the icon for this dialog. The framework does this automatically
 // when the application's main window is not a dialog
 SetIcon(m_hIcon, TRUE); // Set big icon
 SetIcon(m_hIcon, FALSE); // Set small icon

 // TODO: Add extra initialization here
 m_list.InsertColumn(0, "员工编号");
 m_list.SetColumnWidth(0, 100);
 m_list.InsertColumn(1, "员工姓名");
 m_list.SetColumnWidth(1, 100);
 m_list.InsertColumn(2, "员工部门");
 m_list.SetColumnWidth(2, 100);

 personinfo[123].isuser = true;
 strcpy(personinfo[123].id, "123");
 strcpy(personinfo[123].name, "张三");

```

```
strcpy(personinfo[123].dept,"C#");

personinfo[001].isuser = true;
strcpy(personinfo[001].id,"001");
strcpy(personinfo[001].name,"李四");
strcpy(personinfo[001].dept,"软件");
UpdateList();

m_list.SetExtendedStyle(LVS_EX_FULLROWSELECT |
 LVS_EX_GRIDLINES | LVS_EX_FLATSB);//修改列表视图样式
m_id.SetLimitText(3);

IdentifyType = 0;
return TRUE; // return TRUE unless you set the focus to a control
}
```

(4) 将 Biokey.ocx 控件复制到 Win32 目录下并使用 Regsvr32 进行注册。在资源窗体上通过鼠标右键选择 insert activex control 命令添加 Biokey.ocx 控件。

(5) 定义 OnAddRow 方法用来添加新的人员信息。实现代码如下：

```
void CFingerIdentifyDlg::OnAddRow()
{
 CString id,name,dept;
 m_id.GetWindowText(id);
 m_name.GetWindowText(name);
 m_dept.GetWindowText(dept);
 if (id.IsEmpty() || name.IsEmpty() || dept.IsEmpty())
 {
 MessageBox("人员信息不能为空！");
 return;
 }
 int index = atoi(id);
 strcpy(personinfo[index].id,id);
 strcpy(personinfo[index].name,name);
 strcpy(personinfo[index].dept,dept);
 personinfo[index].isuser = true;
 MessageBox("信息添加成功！");
 UpdateList();
}
```

(6) 定义 UpdateList 方法用来实现列表数据的更新。实现代码如下：

```
void CFingerIdentifyDlg::UpdateList()
{
 m_list.DeleteAllItems();
 for (int i = 0 ; i < 999 ; i++)
 {
 if (personinfo[i].isuser)
 {
 int index = m_list.GetItemCount();
 m_list.InsertItem(index,(char *)personinfo[i].id);
 m_list.SetItemText(index,1,(char *)personinfo[i].name);
 m_list.SetItemText(index,2,(char *)personinfo[i].dept);
 }
 }
}
```

(7) 定义 OnRegister 方法用来对指纹进行注册。实现代码如下：

```
void CFingerIdentifyDlg::OnRegister()
{
 CString id;
 m_id.GetWindowText(id);
 if (id.IsEmpty())
 {
 MessageBox("请选择注册人员！");
 return;
 }
 m_zkeng.BeginEnroll();
 MessageBox("开始注册！");
}
```

(8) 定义 OnImageReceived 事件，当采集指纹时将采集的图片绘制在指定的控件中。实现代码如下：

```
void CFingerIdentifyDlg::OnImageReceived(BOOL FAR* AImageValid)
{
 CDC *dc = m_image.GetDC();
 CRect rect;
 m_image.GetClientRect(&rect);//获取图片显示控件大小
 m_zkeng.PrintImageAt((long)dc->m_hDC,0,0,rect.Width(),rect.Height());//将指纹图片绘制在控件中
}
```



(9) 定义 OnFeatureInfo 事件, 用来获取并提示指纹注册时需要采集的次数。实现代码如下:

```
void CFingerIdentifyDlg::OnFeatureInfo(long AQuality)
{
 int times;
 if (m_zkeng.GetIsRegister())//是否正在注册
 {
 times = m_zkeng.GetEnrollIndex() - 1;//获取注册次数
 CString temp;
 ltoa(times,temp.GetBuffer(0),10);
 MessageBox("指纹注册还需要按"+temp+"次! ");
 }
}
```

(10) 定义 OnEnroll 事件, 用来判断指纹注册是否成功, 当成功时将指纹数据存入系统。实现代码如下:

```
void CFingerIdentifyDlg::OnEnroll(BOOL ActionResult, const VARIANT FAR& ATemplate)
{
 if (!ActionResult)
 MessageBox("注册失败! ");
 else
 {
 CString temp;
 m_id.GetWindowText(temp);
 int index = atoi(temp);
 CString sRegTemplate = m_zkeng.GetTemplateAsString();
 strcpy(personinfo[index].finger,sRegTemplate);
 MessageBox("注册成功! ");
 }
}
```

(11) 定义 OnCapture 事件, 来实现指纹比对操作, 当类型为 1 时只比对当前选中的人员, 类型为 2 时比对系统中存在的所有人员。实现代码如下:

```
void CFingerIdentifyDlg::OnCapture(BOOL ActionResult, const VARIANT FAR& ATemplate)
{
 if (IdentifyType == 1)//1:1对比
 {
 CString temp1,temp2;
 int index = atoi(m_list.GetItemText(m_list.GetSelectionMark(),0));
 temp1 = personinfo[index].finger;
 temp2 = m_zkeng.GetTemplateAsString();
 BOOL RegChanged;
 BSTR btemp = temp1.AllocSysString();
 if (m_zkeng.VerFingerFromStr(&btemp,(LPCTSTR)temp2,false,&RegChanged))
 MessageBox("比对成功! ");
 else
 MessageBox("比对失败! ");
 }
 else if (IdentifyType == 2)//1:N对比
 {
 CString temp1,temp2;
 temp2 = m_zkeng.GetTemplateAsString();
 BOOL RegChanged;
 bool isok = false;
 for (int i = 0 ; i < 999 ; i++)
 {
 if (personinfo[i].isuser == false)
 continue;
 temp1 = personinfo[i].finger;
 BSTR btemp = temp1.AllocSysString();
 if (m_zkeng.VerFingerFromStr(&btemp,(LPCTSTR)temp2,false,&RegChanged))
 isok = true;
 }
 if (isok)
 MessageBox("比对成功! ");
 else
 MessageBox("比对失败! ");
 }

 if (IdentifyType != 0)
 IdentifyType = 0;
}
```

## 举一反三

根据本实例, 读者可以:

- 使用指纹头制作考勤管理系统。

## 实例 349 游戏杆控制

这是一个可以提高基础性能的实例

实例位置：光盘\mingrisoft\11\349

### 实例说明

许多游戏的开发都是使用 DirectX 来实现的，特别是使用游戏杆来控制的游戏。DirectX 是一套为 Windows 程序提供对系统硬件更密切控制的组件，通过 DirectX 可以实现许多意想不到的效果。运行程序，通过鼠标或游戏杆可以控制小球的运动。实例运行结果如图 11.30 所示。

### 技术要点

游戏杆的控制是通过 DirectX 中 DirectInput 对象来实现的，此对象由 DirectInput、DirectInputDevice 和 DirectInputEffect 对象组成。在本实例中只使用前两个对象，对象说明如下。

- DirectInput：封装高层 DirectInput 功能，列举设备并用来创建 DirectInputDevice 对象。
- DirectInputDevice：与物理输入设备的接口，例如游戏杆，包括收集和设置设备状态信息的接口，并且用来创建 DirectInputEffect 对象(对于力反馈设备)。
- DirectInputEffect：封装能够在力反馈设备上“播放”的简单效果，提供启动、停止和设置力反馈效果等功能。

### 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在工程中添加名为 Dx.h 的头文件，在此头文件中定义游戏控制所需要的函数头。头文件的定义如下：

```
BOOL InitDirectX(void);
void UnInitDirectX(void);
BOOL GetJoystickInput(int *nX, int *nY, BOOL *bButton);
BOOL GetKeyboardInput(int *nX, int *nY, BOOL *bButton);
```

(3) 在工程中添加名为 Dx.cpp 的文件，此文件实现游戏控制函数。

引入头文件并定义接口变量，实现代码如下：

```
#include "stdafx.h"
#include <windows.h>
#include <Dinput.h>
```

```
LPDIRECTINPUT g_lpDInput = NULL ;
LPDIRECTINPUTDEVICE g_lpDIDeviceKeyboard = NULL;
LPDIRECTINPUTDEVICE2 g_lpDIDeviceJoystick = NULL ;
```

枚举游戏控制设备，并初始化可使用设备的控制范围属性。实现代码如下：

```
BOOL CALLBACK EnumJoy(LPCDIDEVICEINSTANCE lpddi, LPVOID pvRef)
```

```
{
 BOOL bEnumForce ;
 DIPROP_RANGE dipRange ;
 DIPROPDWORD dipDWORD ;
 LPDIRECTINPUTDEVICE lpDIDeviceJoystickTemp ;
 HRESULT hr ;
```

```
bEnumForce = (BOOL) pvRef ;
```

```
//创建游戏杆设备
```

```
if (FAILED(g_lpDInput->CreateDevice(lpddi->guidInstance,
```

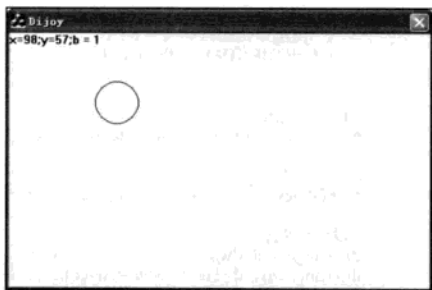


图 11.30 游戏杆控制

```

 &lpDIDeviceJoystickTemp, NULL)))
 {
 g_lpDIDeviceJoystick = NULL;
 return DIENUM_CONTINUE;
 }

 //获取IDirectInputDevice2接口
 hr = lpDIDeviceJoystickTemp->QueryInterface(IID_IDirectInputDevice2,
 (void **)&g_lpDIDeviceJoystick);

 lpDIDeviceJoystickTemp->Release(); //释放接口
 if(FAILED(hr))
 {
 g_lpDIDeviceJoystick = NULL;
 return DIENUM_CONTINUE;
 }

 //设置初始信息
 g_lpDIDeviceJoystick->SetDataFormat(&c_dfDIJoystick);

 //设置控制级别
 g_lpDIDeviceJoystick->SetCooperativeLevel(AfxGetApp()->GetMainWnd()->m_hWnd,
 DISCL_EXCLUSIVE|DISCL_BACKGROUND);

 //设置游戏杆轴范围x方向属性
 dipRange.diph.dwSize = sizeof(dipRange);
 dipRange.diph.dwHeaderSize = sizeof(dipRange.diph);
 dipRange.diph.dwObj = DIJOFS_X;
 dipRange.diph.dwHow = DIPH_BYOFFSET;
 dipRange.lMin = -100;
 dipRange.lMax = +100;
 g_lpDIDeviceJoystick->SetProperty(DIPROP_RANGE, &dipRange.diph);

 //设置游戏杆轴范围y方向属性
 dipRange.diph.dwSize = sizeof(dipRange);
 dipRange.diph.dwHeaderSize = sizeof(dipRange.diph);
 dipRange.diph.dwObj = DIJOFS_Y;
 dipRange.diph.dwHow = DIPH_BYOFFSET;
 dipRange.lMin = -100;
 dipRange.lMax = +100;
 g_lpDIDeviceJoystick->SetProperty(DIPROP_RANGE, &dipRange.diph);

 //设置游戏杆死区范围x方向属性
 dipDWORD.diph.dwSize = sizeof(dipDWORD);
 dipDWORD.diph.dwHeaderSize = sizeof(dipDWORD.diph);
 dipDWORD.diph.dwObj = DIJOFS_X;
 dipDWORD.diph.dwHow = DIPH_BYOFFSET;
 dipDWORD.dwData = 2000;
 g_lpDIDeviceJoystick->SetProperty(DIPROP_DEADZONE, &dipDWORD.diph);

 //设置游戏杆死区范围y方向属性
 dipDWORD.diph.dwSize = sizeof(dipDWORD);
 dipDWORD.diph.dwHeaderSize = sizeof(dipDWORD.diph);
 dipDWORD.diph.dwObj = DIJOFS_Y;
 dipDWORD.diph.dwHow = DIPH_BYOFFSET;
 dipDWORD.dwData = 2000;
 g_lpDIDeviceJoystick->SetProperty(DIPROP_DEADZONE, &dipDWORD.diph);

 return DIENUM_STOP;
}

```

初始化 IDirectInput 接口，并初始化设备。实现代码如下：

```

BOOL InitDirectInput(void)
{
 //创建IDIRECTINPUT接口
 if(FAILED(CoCreateInstance(CLSID_DirectInput, NULL, CLSCTX_INPROC_SERVER,
 IID_IDirectInput, (void **)&g_lpDInput)))
 return FALSE;
 if(FAILED(g_lpDInput->Initialize(AfxGetApp()->m_hInstance,DIRECTINPUT_VERSION)))
 return FALSE;

 //创建键盘设备
 if(FAILED(g_lpDInput->CreateDevice(GUID_SysKeyboard,
 &g_lpDIDeviceKeyboard, NULL)))
 {
 g_lpDIDeviceKeyboard->SetDataFormat(&c_dfDIKeyboard);
 g_lpDIDeviceKeyboard->SetCooperativeLevel(AfxGetApp()->
 GetMainWnd()->m_hWnd,DISCL_FOREGROUND|DISCL_NONEXCLUSIVE);
 }else
 g_lpDIDeviceKeyboard = NULL;

 //获取游戏杆设备信息
 g_lpDInput->EnumDevices(DIDEVTYPE_JOYSTICK, EnumJoy, (LPVOID)TRUE,
 DIEDFL_FORCEFEEDBACK|DIEDFL_ATTACHEDONLY);
}

```

```

 if(!g_lpDIDeviceJoystick)
 g_lpDInput->EnumDevices(DIDEVTYPE_JOYSTICK, EnumJoy, (LPVOID)FALSE,
 DIEDFL_ATTACHEDONLY);

 return true;
 }

```

初始化 COM 对象，并设用 InitDirectInput 函数初始化游戏控制设备。实现代码如下：

```

//初始化DirectX
BOOL InitDirectX(void)
{
 // Initialize COM for this process
 if (FAILED(CoInitialize(NULL)))
 return FALSE;

 // Initialize Direct Input -- DXInput.h
 if(!InitDirectInput())
 return FALSE;

 return TRUE;
}

```

获取键盘信息控制游戏的当前运动参数，实现代码如下：

```

BOOL GetKeyboardInput(int *nX, int *nY, BOOL *bButton)
{
 char cBuffer[256];
 HRESULT hr;

 if(!g_lpDIDeviceKeyboard)//键盘设备存在
 return FALSE;

 //*nX = 0; *nY = 0
 *bButton = FALSE;
 //获取键值
 hr = g_lpDIDeviceKeyboard->GetDeviceState(sizeof(cBuffer),(LPVOID)cBuffer);
 if(hr== DIERR_INPUTLOST || hr== DIERR_NOTACQUIRED)
 {
 g_lpDIDeviceKeyboard->Acquire();
 hr = g_lpDIDeviceKeyboard->GetDeviceState(sizeof(cBuffer),
 (LPVOID)cBuffer);
 }

 if(FAILED(hr))
 return FALSE;

 if(cBuffer[DIK_RIGHT]&0x80)//右键
 (*nX)++;

 if(cBuffer[DIK_LEFT]&0x80)//左键
 (*nX)--;

 if(cBuffer[DIK_UP]&0x80)//向上
 (*nY)--;

 if(cBuffer[DIK_DOWN]&0x80)//向下
 (*nY)++;

 if(cBuffer[DIK_LCONTROL]&0x80)//<CTRL>键
 *bButton = TRUE;

 return TRUE;
}

```

获取游戏杆信息控制游戏的当前运动参数，实现代码如下：

```

BOOL GetJoystickInput(int *nX, int *nY, BOOL *bButton)
{
 HRESULT hr;
 DIJOYSTATE diJoyState;

 if(!g_lpDIDeviceJoystick)//游戏杆
 return FALSE;

 //*nX = 0; *nY = 0;
 *bButton = FALSE;

 hr = g_lpDIDeviceJoystick->Poll(); //检测游戏杆
 if(hr== DIERR_INPUTLOST || hr== DIERR_NOTACQUIRED)
 {

```



```

 g_lpDIDeviceJoystick->Acquire() ;//取得访问权限

 hr = g_lpDIDeviceJoystick->Poll();
 }
 if(FAILED(hr))
 return FALSE;
 //获取游戏杆信息
 if(FAILED(g_lpDIDeviceJoystick->GetDeviceState(sizeof(DIJOYSTATE), &
 diJoyState)))
 return FALSE;

 *nX += diJoyState.IX;
 // *nY = diJoyState.IY
 *bButton = diJoyState.rgbButtons[0];

 return TRUE;
}

```

释放游戏中所使用的 DirectX 接口对象, 实现代码如下:

```

void UnInitDirectInput(void)
{
 if(g_lpDIDeviceKeyboard)//释放键盘设备接口
 {
 g_lpDIDeviceKeyboard->Unacquire();
 g_lpDIDeviceKeyboard->Release();
 }

 if(g_lpDIDeviceJoystick)//释放游戏杆设备接口
 {
 g_lpDIDeviceJoystick->Unacquire();
 g_lpDIDeviceJoystick->Release();
 }

 if(g_lpDInput)//释放控制设备
 g_lpDInput->Release();

 return;
}

void UnInitDirectX(void)
{
 UnInitDirectInput();

 CoUninitialize();
}

```

(4) 在窗体的主窗口的实现文件中引入 Dx.h 头文件, 创建 WM\_TIMER 消息事件实时获取游戏控制信息并绘制小球在当前窗体中的位置。实现代码如下:

```

void CDijoyDlg::OnTimer(UINT nIDEvent)
{
 int x,y,b;
 x = nX;
 y = nY;
 b = bButton;
 if (GetJoystickInput(&nX,&nY,&bButton))//获取游戏杆信息
 {
 this->Invalidate(); //重画
 if (nX != x || bButton != b)
 return;
 }
 nX = x;
 nY = y;
 bButton = b;
 if (GetKeyboardInput(&nX,&nY,&bButton))//获取键盘信息
 {
 this->Invalidate(); //重画
 }
}

CDialog::OnTimer(nIDEvent);
}

```

### 举一反三

根据本实例, 读者可以:

- 通过游戏杆制作汽车运动游戏。

# 第 12 章

## 网络开发技术



- 获取计算机信息
- 局域网控制与管理
- 局域网资源管理
- 网上资源共享
- 套接字应用
- 其他

Visual C++

## 12.1 获取计算机信息

本节主要介绍在局域网中如何获得计算机名称和工作组、如何获取计算机 IP、MAC 地址和端口状态。

### 实例 350 获取计算机名称和工作组

本实例可以提高计算机安全性

实例位置: 光盘\mingrisoft\12\350

#### 实例说明

在局域网中, 可以设置不同的工作组, 通过本例的学习, 读者可以掌握如何获取工作组的名称。运行本例, 计算机名显示在编辑框中, 局域网内的工作组显示在列表中, 如图 12.1 所示。

#### 技术要点

本实例主要使用了 Windows API 函数库中的 WnetOpenEnum、WnetEnumResource 和 WnetCloseEnum 函数。各函数功能如下。

WnetOpenEnum 函数启动对网络资源进行枚举的过程。函数原型如下:

```
DWORD WnetOpenEnum(DWORD dwScope, DWORD dwType, DWORD dwUsage, LPNETRESOURCE lpNetResource, LPHANDLE lphEnum);
```

参数说明:

- dwScope: 指明了此次网络枚举的范围。
- dwType: 指定此次枚举的资源类型。
- dwUsage: 指定枚举资源的用法。
- lpNetResource: 指针类型, 指向特定的数据结构 NETRESOURCE。
- lphEnum: 指针指向存储枚举过程句柄的变量。

WnetEnumResource 函数枚举网络资源。函数原型如下:

```
DWORD WnetEnumResource(HANDLE hEnum, LPDWORD lpcCount, LPVOID lpBuffer, LPDWORD lpBufferSize);
```

参数说明:

- hEnum: 由 WnetOpenEnum 函数的参数 lphEnum 传入。
- lpcCount: 用来决定获取的资源数目最大值。
- lpBuffer: 指针类型, 指向枚举结果存放的缓冲区地址。
- lpBufferSize: 指针类型, 指向枚举结果存储缓冲区大小的变量地址。

WnetCloseEnum 函数结束一次枚举操作。函数原型如下:

```
DWORD WnetCloseEnum(HANDLE hEnum);
```

参数说明:

- hEnum: 该参数由 WnetOpenEnum 函数的参数 lphEnum 传入。

注意: 在使用这些函数之前, 需要初始化向程序中导入 mpr.lib 库和头文件 winnetwk.h。

#### 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“获取计算机名称和工作组”。
- (2) 在窗体上添加一个文本编辑控件和一个列表视图控件。

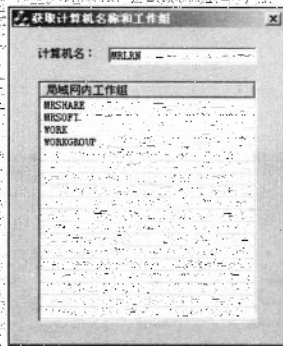


图 12.1 获取计算机名称和工作组



## (3) 主要程序代码。

```

//设置控件的扩展风格
m_grid.SetExtendedStyle(LVS_EX_FLATSB
 [LVS_EX_FULLROWSELECT
 [LVS_EX_HEADERDRAGDROP
 [LVS_EX_ONECLICKACTIVATE
 [LVS_EX_GRIDLINES);
m_grid.InsertColumn(0,"局域网内工作组",LVCFMT_LEFT,220,0); //设置列标题
DWORD nSize = MAX_COMPUTERNAME_LENGTH + 1;
char Buffer[MAX_COMPUTERNAME_LENGTH + 1];
GetComputerName(Buffer,&nSize); //获得机器名
m_edit.SetWindowText(Buffer);
DWORD Count=0xFFFFFFFF,BufSize=4096,Res;
NETRESOURCE* nRes;
HANDLE lphEnum;
LPVOID Buf = new char[4096];
LPVOID Bufwg = new char[4096];

Res = WNetOpenEnum(RESOURCE_GLOBALNET, RESOURCETYPE_ANY,
 RESOURCEUSAGE_CONTAINER,NULL,&lphEnum); //启动网络资源进行枚举
Res=WNetEnumResource(lphEnum,&Count,Buf,&BufSize); //获得枚举的网络资源
nRes=(NETRESOURCE*)Buf;
//通过循环枚举当前资源的下一层资源
for(DWORD n=0;n<Count;n++,nRes++)
{
 DWORD NUM= 0xFFFFFFFF;
 Res = WNetOpenEnum(RESOURCE_GLOBALNET, RESOURCETYPE_ANY, 0,nRes,&lphEnum);
 Res=WNetEnumResource(lphEnum,&NUM,Bufwg,&BufSize);
 int num= BufSize/sizeof(NETRESOURCE);
 nRes=(NETRESOURCE*)Bufwg;
 for(DWORD i=0;i<NUM;i++,nRes++)
 {
 m_grid.InsertItem(i,0);
 m_grid.SetItemText(i,0,nRes->lpRemoteName);
 }
}
delete Buf;
delete Bufwg;
WNetCloseEnum(lphEnum);

```

## 举一反三

根据本实例，读者可以：

- 获取工作组和相关工作组内的计算机名称；
- 监控网络工作组信息。

## 实例 351 通过计算机名获取 IP 地址

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\12\351

## 实例说明

IP 地址能够标识网络中惟一的一台计算机，目前的 IP 地址是 32 位，它被划分为 4 个字节，字节与字节之间用“.”分割，每字节为 8 位，通常用于十进制来表示，例如：F27.0.0.1。当网络中有相同 IP 地址的计算机时系统会提示冲突。每台计算机都有一个惟一的名称，所以，计算机名和 IP 地址是对应的。本例将通过计算机名来获取其 IP 地址，实例效果如图 12.2 所示。

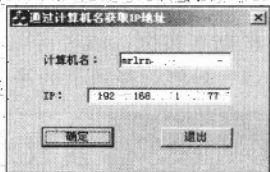


图 12.2 通过计算机名获取 IP 地址

## 技术要点

gethostbyname 是一个在 winsock 单元中声明的函数，该函数能够通过计算机的名称返回其网络信息，这个信息中包括 IP 地址。该函数原型如下：




```
struct hostent FAR * gethostbyname (const char FAR * name);
```

参数说明:

- name: 包含计算机名称的字符串。
- 返回值: 该函数返回值为 HOSTENT 结构类型的指针, 该类型声明如下:

```
struct hostent {
 char FAR * h_name;
 char FAR * FAR * h_aliases;
 short h_addrtype;
 short h_length;
 char FAR * FAR * h_addr_list;
};
```

 注意: 在使用该函数之前, 需要导入 ws2\_32.lib 库和头文件 afxsock.h。

## 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“通过计算机名获取 IP 地址”。
- (2) 选择菜单 Project/Add To Project, 然后选择 Components and Controls..., 弹出 Components and Controls Gallery 窗口, 双击窗体中的 Registered ActiveX Controls 文件夹, 找到 Microsoft ADO Data Control6.0(SP4)(OLEDB)选项, 双击它, 取默认值, 添加控件, 单击“Close”按钮, ADO Data 控件就添加到控件面板中了。用同样的方法找到 Microsoft DataGrid Control6.0(SP5)(OLEDB)选项, 添加 DataGrid 控件。
- (3) 在窗体上添加一个文本编辑控件、一个 IP 地址控件和两个按钮控件。
- (4) 在应用程序中初始化套接字。代码如下:

```
WSADATA wsd;
WSAStartup(MAKEWORD(2,2), &wsd);
```

- (5) 通过计算机名获得 IP。代码如下:

```
void CGetIPDlg::OnButok()
{
 CString str = "";
 m_name.GetWindowText(name);
 struct hostent * pHost;
 pHost = gethostbyname(name);
 //格式化IP地址
 for(int i=0; i<4; i++)
 {
 CString addr;
 if(i > 0)
 {
 str += ".";
 }
 addr.Format("%u", (unsigned int)((unsigned char*)pHost->h_addr_list[0])[i]);
 str += addr;
 }
 m_ip.SetWindowText(str);
}
```

## 举一反三

根据本实例, 读者可以:

- 如何获得网卡的序列号。

## 实例 352 获取本机 MAC 地址

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\12\352

## 实例说明

在实际的应用程序中, 经常需要在程序运行时获取 MAC 地址, 用来作为某种标识。MAC

地址是网络适配器的物理地址,网络适配器又称网卡,它是计算机通信的主要设备,在出厂时 MAC 的地址就写入到了适配器中,出现两块相同 MAC 地址的网卡的几率非常的小,几乎就等于 0。所以用 MAC 的地址几乎就能够标识网络中一台惟一的计算机。实例运行结果如图 12.3 所示。

### 技术要点

本实例使用了 Netbios 函数,通过该函数就能够获取本机的 MAC 地址,该函数原型如下:

UCHAR Netbios( PNCB pncb );

参数 pncb 是一个 NCB 类型的结构,该结构定义如下:

```
typedef struct _NCB {
 UCHAR ncb_command;
 UCHAR ncb_retcode;
 UCHAR ncb_lsn;
 UCHAR ncb_num;
 PCHAR ncb_buffer;
 WORD ncb_length;
 UCHAR ncb_callname[NCBNAMSZ];
 UCHAR ncb_name[NCBNAMSZ];
 UCHAR ncb_rto;
 UCHAR ncb_sto;
 void (*ncb_post)(struct _NCB *);
 UCHAR ncb_lana_num;
 UCHAR ncb_cmd_cplt;
 UCHAR ncb_reserve[10];
 HANDLE ncb_event;
} NCB;
```

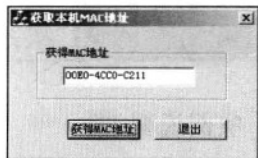


图 12.3 获取本机 MAC 地址

### 实现过程

- (1) 新建一个基于对话框的应用程序,将窗体标题改为“获取本机 MAC 的地址”。
- (2) 在窗体上添加一个文本编辑控件和两个按钮控件。
- (3) 主要程序代码。

```
void CGetMACDlg::OnButmac()
{
 CString strMac;
 NCB ncb;
 ADAPTER_STATUS adapt;
 memset(&ncb,0,sizeof(ncb));
 ncb.ncb_command = NCBRESET; //首先对网卡发送一个NCBRESET命令以便进行初始化
 Netbios(&ncb);
 ncb.ncb_command = NCBASTAT;
 strcpy((char *)ncb.ncb_callname,"*");
 ncb.ncb_buffer = (unsigned char *)&adapt; //指定返回的信息存放的变量
 ncb.ncb_length = sizeof(adapt);
 // 发送NCBASTAT命令以获取网卡的信息
 Netbios(&ncb);
 strMac.Format("%02X%02X-%02X%02X-%02X%02X\n", //把网卡MAC地址格式化成常用的十六进制形式
 adapt.adapter_address[0],
 adapt.adapter_address[1],
 adapt.adapter_address[2],
 adapt.adapter_address[3],
 adapt.adapter_address[4],
 adapt.adapter_address[5]);
 m_edit.SetWindowText(strMac);
}
```

### 举一反三

根据本实例,读者可以:

- 利用网卡地址设计软件注册码。

## 实例 353

## 获得系统打开的端口和状态

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\12\353

## 实例说明

当两台计算机之间进行通信和信息传递时, 每台计算机都需要开启一个端口。这个端口就是与另一台计算机通信的通道。木马或黑客程序都需要一个端口与远程的计算机进行通信, 这些端口是不固定的, 所以很难防范。但是将系统中所有的端口列出察看, 就会知道系统是否正在与网络中的计算机进行通信。本例将列出当前系统中打开的端口和端口信息, 运行结果如图 12.4 所示。

## 技术要点

DOS SHELL 命令 netstat 是一个强大的网络命令。它能够获得系统中的端口信息。其命令参数 -a 表示列出所有的链接和侦听端口。在程序中使用 WinExec 函数可以调用该命令。该函数原型如下:

```
UINT WinExec(LPCSTR lpCmdLine,UINT uCmdShow);
```

参数说明:

- lpCmdLine: 命令行。
- uCmdShow: 显示类型。

调用 netstat 命令时可以使用重定向功能将结果保存到一个文本文件中, 然后在程序中将该文件打开显示, 就能够浏览目前端口信息了。调用的命令如下:

```
Command.com /c netstat -a -n > c:\port.txt
```

## 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“获得系统打开的端口和状态”。
- (2) 在窗体上添加一个格式文本编辑控件和两个按钮控件。
- (3) 主要程序代码。

```
void CHuodeDKDlg::OnInspect()
{
 char buffer[100];
 GetWindowsDirectory(buffer,100);
 strcat(buffer,"\\system32\\netstat -a -n > c:\\port.txt");
 CString strPath;
 strPath.Format("Command.com /c %s",buffer);
 WinExec(strPath,SW_HIDE); //监听端口
 Sleep(5000); //延时
 CString str="";
 char sread[10000];
 CFile mfile(_T("c:\\port.txt"),CFile::modeRead); //打开文件
 mfile.Read(sread,10000); //读取文件内容
 for(int i=0;i<mfile.GetLength();i++)
 {
 str += sread[i];
 }
 m_richedit.SetWindowText(str); //将端口信息显示到控件中
}
```

## 举一反三

根据本实例, 读者可以:

- 得到局域网中两台计算机的端口号;

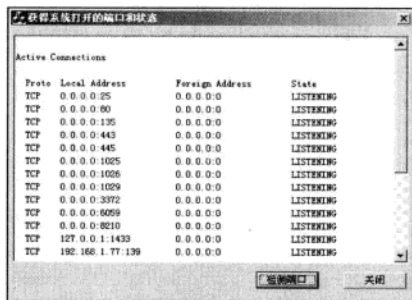


图 12.4 获得系统打开的端口和状态

- 实时监控计算机的端口信息。

## 12.2 局域网控制与管理

局域网的管理是局域网编程中不可缺少的组成部分。网络管理者需要实时掌握局域网内部计算机的连接状态。本节将介绍在局域网中实现获取局域网计算机名称和 IP 以及控制局域网上的计算机方面的知识。

### 实例 354

### 获取局域网计算机名称和 IP

本实例可以提高工作效率


实例位置：光盘\mingrisoft\12\354

#### 实例说明


在本程序运行时，检索整个局域网络，将局域网络中的计算机名和 IP 地址显示在 ListControl 控件中。实例运行结果如图 12.5 所示。

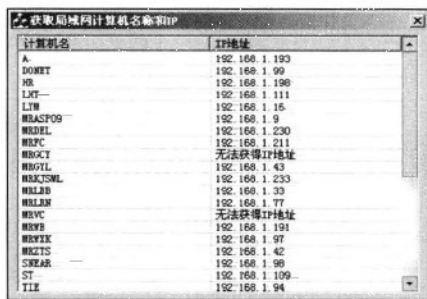
#### 技术要点

本实例主要使用了 Windows API 函数库中的 WnetOpenEnum、WnetEnumResource 和 WnetCloseEnum 函数枚举局域网计算机名。

 注意：在使用这些函数之前，需要初始化向程序中导入 mpr.lib 库和头文件 winnetwk.h。

通过 gethostbyname 函数获取计算机 IP 地址。

 注意：在使用该函数之前，需要导入 ws2\_32.lib 库和头文件 afxsock.h。



| 计算机名    | IP地址          |
|---------|---------------|
| A       | 192.168.1.193 |
| DONET   | 192.168.1.99  |
| HR      | 192.168.1.198 |
| LJT     | 192.168.1.111 |
| LTN     | 192.168.1.15  |
| WEAFTOP | 192.168.1.9   |
| WDEL    | 192.168.1.230 |
| WDFC    | 192.168.1.211 |
| WDFCT   | 无法获得IP地址      |
| WDFL    | 192.168.1.43  |
| WDFCSWL | 192.168.1.233 |
| WDLB    | 192.168.1.33  |
| WDLN    | 192.168.1.77  |
| WDFC    | 无法获得IP地址      |
| WDFB    | 192.168.1.191 |
| WDFX    | 192.168.1.97  |
| WDFTS   | 192.168.1.42  |
| WDFR    | 192.168.1.98  |
| ST      | 192.168.1.109 |
| TLE     | 192.168.1.94  |

图 12.5 获取局域网计算机名称和 IP

#### 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“获取局域网计算机名称和 IP”。
- (2) 在窗体上添加一个列表视图控件。
- (3) 主要程序代码。

```
WSADATA wsd;
WSAStartup(MAKEWORD(2,2),&wsd); //初始化套接字
//设置控件扩展风格
m_grid.SetExtendedStyle(LVS_EX_FLATSB
|LVS_EX_FULLROWSELECT
|LVS_EX_HEADERDRAGDROP
|LVS_EX_ONECLICKACTIVATE
|LVS_EX_GRIDLINES);
//设置列标题
m_grid.InsertColumn(0,"计算机名",LVCFMT_LEFT,200,0);
m_grid.InsertColumn(1,"IP地址",LVCFMT_LEFT,200,0);
DWORD Count=0xFFFFFFFF,Bufsize=4096,Res;
//枚举网络资源
NETRESOURCE* nRes;
NETRESOURCE* nRes1;
NETRESOURCE* nRes2;
HANDLE lphEnum;
LPVOID Buf = new char[4096];
LPVOID Buf1 = new char[4096];
LPVOID Buf2 = new char[4096];
Res = WNetOpenEnum(RESOURCE_GLOBALNET, RESOURCETYPE_ANY, RESOURCEUSAGE_CONTAINER,
NULL,&lphEnum);
Res=WNetEnumResource(lphEnum,&Count,Buf,&Bufsize);
nRes=(NETRESOURCE*)Buf;
```



```

for(DWORD n=0;n<Count;n++,nRes++)
{
 DWORD Count1=0xFFFFFFFF;
 Res = WNetOpenEnum(RESOURCE_GLOBALNET, RESOURCETYPE_ANY, RESOURCEUSAGE_CONTAINER,
nRes,&lphEnum);
 Res=WNetEnumResource(lphEnum,&Count1,Buf1,&Bufsize);
 nRes1=(NETRESOURCE*)Buf1;
 //循环枚举下一层资源
 for(DWORD i=0;i<Count1;i++,nRes1++)
 {
 DWORD Count2=0xFFFFFFFF;
 Res = WNetOpenEnum(RESOURCE_GLOBALNET, RESOURCETYPE_ANY, RESOURCEUSAGE_CONTAINER,
nRes1,&lphEnum);
 Res=WNetEnumResource(lphEnum,&Count2,Buf2,&Bufsize);
 nRes2=(NETRESOURCE*)Buf2;
 for(DWORD j=0;j<Count2;j++,nRes2++)
 {
 m_grid.InsertItem(j,0);
 CString sName=nRes2->lpRemoteName;
 sName=sName.Right(sName.GetLength()-2);
 m_grid.SetItemText(j,0,sName);
 CString str="";
 struct hostent * pHost;
 pHost = gethostbyname(sName);
 if(pHost==NULL)
 {
 m_grid.SetItemText(j,1,"无法获得IP地址");
 }
 else
 {
 for(int n=0;n<4;n++)
 {
 CString addr;
 if(n > 0)
 {
 str += " ";
 }
 addr.Format("%u",((unsigned int)((unsigned char*)pHost->h_addr_list[0])[n]);
 str += addr;
 }
 m_grid.SetItemText(j,1,str);
 }
 }
 }
}
delete Buf;
delete Buf1;
delete Buf2;
WNetCloseEnum(lphEnum);

```

### 举一反三

根据本实例，读者可以：

- 根据用户的 IP 地址获得该 IP 地址的计算机名；
- 获得局域网中的资源情况。

## 实例 355 远程控制局域网计算机

本实例是一个方便操作、提高效率的程序

实例位置：光盘\mingrisoft\12\355

### 实例说明

网上的许多木马、黑客程序一般都具有远程控制计算机的能力。例如，它能够让你的计算机定时关机、窃取文件信息等。它是如何实现远程控制的呢？本例中笔者设计了远程控制计算机的程序，运行结果如图 12.6 所示。

## 技术要点

要实现远程控制计算机的功能，软件应采用客户/服务器模式设计，即设计两个应用程序，一个作为客户端应用程序，一个作为服务器端应用程序。当客户端应用程序打开时，服务器端的应用程序就可以与客户端进行通信了。一些黑客程序就是通过服务器端向客户端发送指令，在客户端执行非法操作的。

本例利用 Windows Socket API 函数开发一个基于 TCP 的网络应用程序。服务器端应用程序首先需要设置本机的地址和端口号，然后利用套接字绑定地址，并开始监听客户端，如果有客户端连接，接受其连接请求，并发送信息到客户端。客户端应用程序需要设置连接的服务器信息，包括服务器地址和端口号，然后连接服务器，在服务器接受连接请求后，向服务器发送数据。

## 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“远程控制局域网计算机（客户端）”。
- (2) 在窗体上添加 1 个文本编辑控件和 5 个按钮控件。
- (3) 主要程序代码。

```
//连接服务器
void CClientDlg::LianJie()
{
 UpdateData(true);
 CString str="";
 struct hostent * pHost;
 pHost = gethostbyname(m_name); //获得IP地址
 if(pHost==NULL)
 {
 MessageBox("无法获得IP地址");
 return;
 }
 else
 {
 //格式化IP地址
 for(int n=0;n<4;n++)
 {
 CString addr;
 if(n > 0)
 {
 str += ".";
 }
 addr.Format("%u",(unsigned int)((unsigned char*)pHost->h_addr_list[0])[n]);
 str += addr;
 }
 }
 //服务器端地址
 sockaddr_in serveraddr;
 serveraddr.sin_family = AF_INET;
 serveraddr.sin_port = htons(70);
 serveraddr.sin_addr.S_un.S_addr = inet_addr(str);
 connect(m_client,(sockaddr*)&serveraddr,sizeof(serveraddr));
 WSASyncSelect(m_client,m_hWnd,1000,FD_READ);
}
```

- (4) 新建一个基于对话框的应用程序，将窗体标题改为“远程控制局域网计算机（服务器端）”。
- (5) 主要程序代码。监听客户端，代码如下：

```
//创建套接字
m_server = socket(AF_INET,SOCK_STREAM,0);
//将网络中的事件关联到窗口的消息函数中
WSASyncSelect(m_server,m_hWnd,20000,FD_WRITE|FD_READ|FD_ACCEPT);
m_client = 0;
m_serverIP = "";
DWORD nSize = MAX_COMPUTERNAME_LENGTH + 1;
```

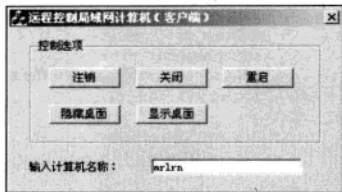


图 12.6 远程控制局域网计算机

```
char Buffer[MAX_COMPUTERNAME_LENGTH + 1];
GetComputerName(Buffer,&nSize);
CString str="",name;
struct hostent * pHost;
pHost = gethostbyname(Buffer);
for(int i=0;i<4;i++)
{
 CString addr;
 if(i > 0)
 {
 str += ".";
 }
 addr.Format("%u",(unsigned int)((unsigned char*)pHost->h_addr_list[0])[i]);
 str += addr;
}

//服务器端地址
sockaddr_in serveraddr;
serveraddr.sin_family = AF_INET;
m_serverIP = str;
//设置本机地址
serveraddr.sin_addr.S_un.S_addr = inet_addr(m_serverIP);
UpdateData(TRUE);
//设置端口号
serveraddr.sin_port = htons(70);
//绑定地址
if (bind(m_server,(sockaddr*)&serveraddr,sizeof(serveraddr)))
{
 MessageBox("绑定地址失败。");
 return;
}
//开始监听
listen(m_server,5);
```

#### (6) 响应客户端按钮，代码如下：

```
sockaddr_in serveraddr;
char buffer[1];
int len = sizeof(serveraddr);
if (m_client==0) //客户端连接服务器
{
 m_client = accept(m_server,(struct sockaddr*)&serveraddr,&len);
}
else
{
 //接收客户端的数据
 int num = recv(m_client,buffer,1,0);
 CWnd* cwnd;
 static HANDLE hToken;
 static TOKEN_PRIVILEGES tp;
 static LUID luid;
 OpenProcessToken(GetCurrentProcess(),TOKEN_ADJUST_PRIVILEGES|TOKEN_QUERY,&hToken);
 LookupPrivilegeValue(NULL,SE_SHUTDOWN_NAME,&luid);
 tp.PrivilegeCount = 1;
 tp.Privileges [0].Luid =luid;
 tp.Privileges [0].Attributes =SE_PRIVILEGE_ENABLED;
 AdjustTokenPrivileges(hToken,FALSE,&tp,sizeof(TOKEN_PRIVILEGES),NULL, NULL);
 switch(buffer[0])
 {
 case '1':
 cwnd = FindWindow("ProgMan",NULL);
 ::ShowWindow(cwnd->GetSafeHwnd(),SW_HIDE);
 break;
 case '2':
 cwnd = FindWindow("ProgMan",NULL);
 ::ShowWindow(cwnd->GetSafeHwnd(),SW_SHOW);
 break;
 case '3':
 ::ExitWindowsEx(EWX_LOGOFF,0);
 break;
 case '4':
 ::ExitWindowsEx(EWX_SHUTDOWN,0);
 break;
 case '5':
 ::ExitWindowsEx(EWX_REBOOT,0);
 break;
 }
}
```

## 举一反三

根据本实例，读者可以：

- 开发网络通信程序。

## 12.3 局域网资源管理

局域网的管理是局域网编程中不可缺少的组成部分。局域网管理者必须利用监测程序监测局域网内部所有的计算机。同时，为保证局域网的稳定，监测程序又不能占用过多的系统资源。本节将介绍在局域网中如何实现将局域网资源保存到数据库中以及如何如何进行局域网屏幕监控。

## 实例 356 计算机监控

本实例是一个方便操作、提高效率的程序

实例位置：光盘\mingrisoft\12\356

## 实例说明

在目前许多网络软件都具有远程控制的功能，而计算机监控就是其中的部分功能之一。该实例所实现的功能就是将服务器所在计算机的屏幕活动返应到客户端程序的窗口中。实例运行效果如图 12.7 所示。

## 技术要点

该实例是屏幕监控软件，也就是说将服务器端的屏目显示图片发送到客户端再显示在窗口中。实现这样的操作存在两种方法：一种是将图片信息存入文件，再将文件发送到客户端；另一种是将图片信息存入内存，直接以内存数据发送到客户端。该实例使用的是第二种方法。

图片信息通常由两部分组成：一是 BITMAP 结构信息，二是图片数据。BITMAP 结构可以通过 CBitmap 类直接获取，实现代码如下：

```
CDC dc,bmpdc;
int width,height;
dc.CreateDC("DISPLAY",NULL,NULL,NULL); //创建关联屏幕的画布
CBitmap bm; //位图类
width = GetSystemMetrics(SM_CXSCREEN); //屏幕宽度
height = GetSystemMetrics(SM_CYSCREEN); //屏幕高度
bm.CreateCompatibleBitmap(&dc,
 width,
 height); //创建与屏幕画布兼容的位图
bmpdc.CreateCompatibleDC(&dc); //创建与屏幕画布兼容的临时画布
bmpdc.SelectObject(&bm); //选择位图
bmpdc.BitBlt(0,0,width,height,&dc,0,0,SRCCOPY); //将屏幕图像复制到临时画布中
bm.GetBitmap(&bitmap);//获取位图结构
```

而在获取图片数据时首先要设置 BITMAPINFOHEADER 结构信息，再利用 GetDIBits 函数通过 BITMAPINFOHEADER 结构获取图片数据。实现代码如下：

```
bm.GetBitmap(&bitmap); //获取位图结构
size = bitmap.bmWidthBytes * bitmap.bmHeight; //计算图片数据大小
bmpdata = new char[size]; //定义图片数据存储区

BITMAPINFOHEADER bih; //位图信息头结构
bih.biBitCount=bitmap.bmBitsPixel; //图片像素位数
bih.biClrImportant=0;
bih.biClrUsed=0;
```

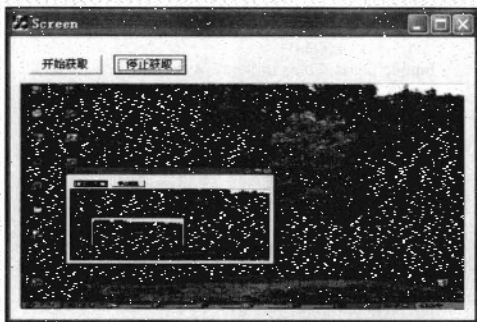


图 12.7 计算机监控



```

bih.biCompression=0;
bih.biHeight=bitmap.bmHeight; //图片高度
bih.biPlanes=1;
bih.biSize=sizeof(BITMAPINFOHEADER);
bih.biSizeImage=size; //图片大小
bih.biWidth=bitmap.bmWidth; //图片宽度
bih.biXPelsPerMeter=0;
bih.biYPelsPerMeter=0;
//获取图片数据
GetDIBits(dc,bm,0,bih.biHeight,bmpdata,(BITMAPINFO*)&bih,DIB_RGB_COLORS);

```

在客户端，也就是图片数据的接收端会根据接收到的 BITMAP 结构和图片数据通过 SetDIBits 方法生成图片。实现代码如下：

```
SetDIBits(bmpdc,m_hDC,bm,0,bitmap.bmHeight,bmpdata,(BITMAPINFO*)&bih,DIB_RGB_COLORS);
```

## 实现过程

(1) 新建一个基于对话框的工程，并命名为 Server (服务器端)。

(2) 在窗体类中定义 GetScreen 方法来获取服务器端屏幕图像及发送所需要的 BITMAP 结构和位图数据。实现代码如下：

```

void CServerDlg::GetScreen()
{
CDC dc,bmpdc; //屏幕画布与临时画布
int width,height; //屏幕的宽度和高度
dc.CreateDC("DISPLAY",NULL,NULL,NULL); //根据屏幕上下文创建画布
CBitmap bm; //定义存储屏幕图像的位图
width = GetSystemMetrics(SM_CXSCREEN); //屏幕宽度
height = GetSystemMetrics(SM_CYSCREEN); //屏幕高度
bm.CreateCompatibleBitmap(&dc,
 width,
 height); //创建与屏幕兼容的位图
bmpdc.CreateCompatibleDC(&dc); //创建与屏幕画布兼容的临时画布
bmpdc.SelectObject(&bm); //选择图片
bmpdc.BitBlt(0,0,width,height,&dc,0,0,SRCCOPY); //将屏幕图像复制到位图中
bm.GetBitmap(&bitmap); //获取位图结构
size = bitmap.bmWidthBytes * bitmap.bmHeight; //计算位图数据大小
bmpdata = new char[size]; //创建存储位图数据的缓冲区

BITMAPINFOHEADER bih; //位图信息头
bih.biBitCount=bitmap.bmBitsPixel; //颜色位数
bih.biClrImportant=0;
bih.biClrUsed=0;
bih.biCompression=0;
bih.biHeight=bitmap.bmHeight; //位图高度
bih.biPlanes=1;
bih.biSize=sizeof(BITMAPINFOHEADER);
bih.biSizeImage=size; //位图大小
bih.biWidth=bitmap.bmWidth; //位图宽度
bih.biXPelsPerMeter=0;
bih.biYPelsPerMeter=0;
//获取位图数据到bmpdata
GetDIBits(dc,bm,0,bih.biHeight,bmpdata,(BITMAPINFO*)&bih,DIB_RGB_COLORS);
}

```

(3) 在窗体类中实现 OnReceive 方法来接收客户端发送的命令并执行，该方法由 CSocket 类的子类 CTCPCClientSocket 来调用。实现代码如下：

```

void CServerDlg::OnReceive(CSocket * socket, int nErrorCode)
{
char buffer[255];
int len = socket->Receive(buffer,255); //接收数据，并返回实现接收数据的长度
buffer[len] = '\0'; //添加结束符
switch (*buffer)
{
case 'D':
 SendBufferData(socket); //发送图像数据
 break;
case 'M':
 SendBitmap(socket); //发送位图结构
 break;
default:
 AfxMessageBox(buffer);
}
}

```

(4) 在 OnReceive 方法中分别调用了 SendBitData 方法和 SendBitmap 方法，用来发送可组成图像的信息。实现代码如下：

```
void CServerDlg::SendBitData(CSocket *socket)
{
 char *data = bmpdata; //指向图片数据的指针
 int sendlen=0; //实际发送长度
 int len = 0; //发送总长度
 do
 {
 sendlen = socket->Send(data,size); //发送数据
 len += sendlen;
 data += sendlen;
 }while(len<size);
 delete bmpdata;
 bmpdata = NULL;
 size = 0;
}

void CServerDlg::SendBitmap(CSocket *socket)
{
 GetScreen(); //获取屏幕信息
 socket->Send(&bitmap,sizeof(BITMAP)); //发送位图结构
}
```

(5) 新建一个基于对话框的项目，命名为 Screen（客户端）。

(6) 在窗体上添加两个按钮控件和一个图片控件。

(7) 在窗体类中实现 GetScreen 方法用于向服务器发送命令，并接收数据。实现代码如下：

```
void CScreenDlg::GetScreen()
{
 char *buffer = "M"; //命令M，获取BITMAP结构
 clientsocket.Send(buffer,strlen(buffer)); //发送命令
 clientsocket.Receive(&bitmap,sizeof(BITMAP)); //接收数据
 size = bitmap.bmWidthBytes * bitmap.bmHeight; //计算图像大小
 bmpdata = new char[size]; //指定存储图像数据的缓冲区
 char *data = bmpdata;
 int len,receivelen;
 len = receivelen = 0;
 buffer = "D"; //命令D，获取图像数据
 clientsocket.Send(buffer,strlen(buffer)); //发送命令
 do
 {
 receivelen = clientsocket.Receive(data,size);
 len += receivelen;
 data += receivelen;
 }while(len<size); //循环接收图像数据
 DrawScreen(); //绘制屏幕图像
 delete bmpdata;
 bmpdata = NULL;
 size = 0;
}
```

(8) 在窗体类中实现 DrawScreen 方法用于根据接收到的图像信息在窗体上的 Picture 控件中绘制图像。实现代码如下：

```
void CScreenDlg::DrawScreen()
{
 CDC *dc = m_drawscreen.GetDC(); //Picture控件画布
 BITMAPINFOHEADER bih; //位图信息头
 bih.biBitCount=bitmap.bmBitsPixel; //颜色位数
 bih.biClrImportant=0;
 bih.biClrUsed=0;
 bih.biCompression=0;
 bih.biHeight=bitmap.bmHeight; //位图高度
 bih.biPlanes=1;
 bih.biSize=sizeof(BITMAPINFOHEADER);
 bih.biSizeImage=size; //位图大小
 bih.biWidth=bitmap.bmWidth; //位图宽度
 bih.biXPelsPerMeter=0;
 bih.biYPelsPerMeter=0;

 CBitmap bm; //根据位图结构创建位图
 bm.CreateBitmapIndirect(&bitmap);
 CDC bmpdc;
 bmpdc.CreateCompatibleDC(dc);
 SetDIBits(bmpdc.m_hDC,bm,0,bitmap.bmHeight,bmpdata,
```

```
(BITMAPINFO*)&bih,DIB_RGB_COLORS); //向位图中添加数据
bmpdc.SelectObject(&bm); //选择位图
CRect rect;
m_drawscreen.GetClientRect(&rect);
//将图像绘制到Picture控件中
dc->StretchBlt(0,0,rect.Width(),rect.Height(),
 &bmpdc,0,0,bitmap.bmWidth,bitmap.bmHeight,SRCCOPY);
}
```

(9) 在窗口中单击“开始获取”按钮或“停止获取”按钮，将开启和停止窗体 OnTimer 消息事件的运行，用来不断的获取图像信息并绘制。实现代码如下：

```
void CScreenDlg::OnStart()
{
 clientsocket.Create(); //创建套接字
 run = false;
 bool ret = clientsocket.Connect("127.0.0.1",1033); //连接
 if (!ret)
 return;
 this->SetTimer(0,1000,NULL); //运行OnTimer消息事件
}
void CScreenDlg::OnStop()
{
 this->KillTimer(0); //停止OnTimer消息事件
 clientsocket.ShutDown(2);
 clientsocket.Close();
}
void CScreenDlg::OnTimer(UINT nIDEvent)
{
 if (nIDEvent != 0)
 return;
 if (run)
 return;
 run = true;
 GetScreen(); //获取远程屏幕信息并绘制
 run = false;
 CDialog::OnTimer(nIDEvent);
}
```

### 举一反三

根据本实例，读者可以：

- 监控并控制客户机屏幕。

## 实例 357 实现进程间通信

本实例可以提高网络安全性

实例位置：光盘\mingrisoft\12\357

### 实例说明

在开发一些监控软件或网络应用程序时，经常需要进行进程间通信，即在程序间实现信息传递。本例将通过消息传递的方式实现进程之间的通信。实例运行结果如图 12.8、图 12.9 所示。

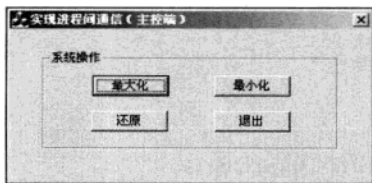


图 12.8 实现进程间的通信（主控端）



图 12.9 实现进程间的通信（被控端）

## 技术要点

Windows 的消息机制允许程序间通过使用 `PostMessage`、`SendMessage` 之类的函数进行相互通信, 程序间可以通过 `WM_USER` 常量加上一个值创建一个消息标识, 然后使用这个消息标识进行相互通信。如果其他程序与发送程序定义了相同的消息标识, 可能会产生预想不到的结果, 基于这个原因, 用户最好通过 `RegisterWindowMessage` 函数注册一个惟一的消息标识用于进程间通信。

从上面的分析可以得出, 实现进程间通信首先需要定义或注册消息, 然后发送消息, 最后由接收端接收并处理消息。在这个过程中, 需要使用 `RegisterWindowMessage` 函数、`PostMessage` 函数来注册和发送消息, 下面分别加以介绍。

(1) `RegisterWindowMessage`。该函数在系统中产生一个惟一的消息标识, 如果两个程序注册了同一个消息字符串, 则两个程序会得到相同的消息标识。函数原型如下:

```
UINT RegisterWindowMessage(LPCTSTR lpString);
```

参数说明:

- `lpString`: 注册消息的字符串, 如果函数执行成功, 返回值是一个新的消息标识, 如果函数执行失败, 返回值为 0。

(2) `PostMessage`。该函数将消息放入拥有指定窗口的线程的消息队列中。函数原型如下:

```
BOOL PostMessage(HWND hWnd,UINT Msg,WPARAM wParam,LPARAM lParam);
```

参数说明:

- `hWnd`: 代表一个窗口句柄, 该窗口的窗口过程将接收特定的消息。
- `Msg`: 发送的消息标识。
- `wParam` 与 `lParam`: 与被发送消息相关的附加信息。如果函数执行成功, 返回值为 `TRUE`, 否则为 `FALSE`。

## 实现过程

(1) 新建一个基于对话框的应用程序, 将窗体标题改为“实现进程间通信 (主控端)”。

(2) 向窗体中添加 4 个按钮控件。

(3) 主要程序代码。

```
//最大化被控程序
void CCourseDlg::OnButmax()
{
 // TODO: Add your control notification handler code here
 UINT maxMsg = RegisterWindowMessage("最大化");
 ::PostMessage(HWND_BROADCAST,maxMsg,0,0);
}
//最小化被控程序
void CCourseDlg::OnButmin()
{
 // TODO: Add your control notification handler code here
 UINT minMsg = RegisterWindowMessage("最小化");
 ::PostMessage(HWND_BROADCAST,minMsg,0,0);
}
//还原被控程序
void CCourseDlg::OnButrevert()
{
 // TODO: Add your control notification handler code here
 UINT revMsg = RegisterWindowMessage("还原");
 ::PostMessage(HWND_BROADCAST,revMsg,0,0);
}
//关闭被控程序
void CCourseDlg::OnButexit()
{
 // TODO: Add your control notification handler code here
 UINT closeMsg = RegisterWindowMessage("关闭");
 ::PostMessage(HWND_BROADCAST,closeMsg,0,0);
}
```

(4) 新建一个基于对话框的应用程序, 将窗体标题改为“实现进程间通信 (被控端)”。



(5) 向窗体中添加一个图片控件,并向资源中导入一幅 BMP 图片。

(6) 主要程序代码。

```
BOOL CCourseSonDlg::PreTranslateMessage(MSG* pMsg)
{
 // TODO: Add your specialized code here and/or call the base class
 if(pMsg->message == minMsg)
 {
 PostMessage(WM_SYSCOMMAND,SC_MINIMIZE,0);
 }
 else if(pMsg->message == maxMsg)
 {
 PostMessage(WM_SYSCOMMAND,SC_MAXIMIZE,0);
 }
 else if(pMsg->message == revMsg)
 {
 PostMessage(WM_SYSCOMMAND,SC_RESTORE,0);
 }
 else if(pMsg->message == closeMsg)
 {
 PostMessage(WM_SYSCOMMAND,SC_CLOSE,0);
 }
 return CDialog::PreTranslateMessage(pMsg);
}
```

### 举一反三

根据本实例,读者可以:

- 监控程序间通信。

## 实例 358

### 利用内存映射实现进程间通信

这是一个可以提高基础性能的实例

实例位置:光盘\mingrisoft\12\358

### 实例说明

在开发程序过程中,经常涉及进程之间的通信。例如,从一个应用程序中访问另一个应用程序中的数据。本实例利用内存映射实现两个应用程序间的数据文本的交换,效果如图 12.10、图 12.11 所示。

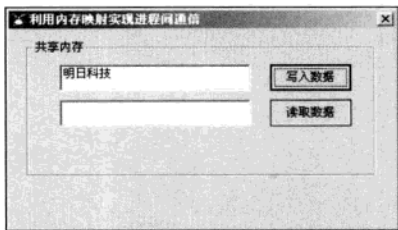


图 12.10 内存映射实例 1

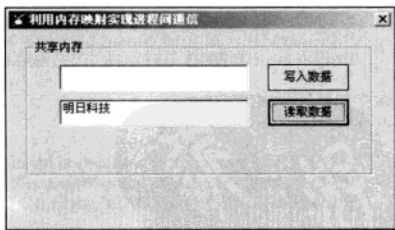


图 12.11 内存映射实例 2

### 技术要点

要实现内存映射,首先需要调用 `CreateFileMapping` 函数创建一个内存映射对象,然后调用 `MapViewOfFile` 函数将内存映射对象映射到进程的地址空间中,这样就可以设置或读取共享内存中的数据了。下面介绍 `CreateFileMapping`、`MapViewOfFile` 函数的使用。

(1) `CreateFileMapping` 函数。该函数用于创建一个内存映射对象,其语法如下:

```
HANDLE CreateFileMapping(HANDLE hFile, LPSECURITY_ATTRIBUTES lpFileMappingAttributes,DWORD flProtect,
DWORD dwMaximumSizeHigh,
DWORD dwMaximumSizeLow,LPCTSTR lpName);
```

参数说明：

- hFile：标识创建映射对象的句柄。
- lpFileMappingAttributes：确定安全属性，即函数返回的句柄能否被子进程继承。
- flProtect：标识映射对象的访问权限，通常为 PAGE\_READWRITE，表示具有读写权限。
- dwMaximumSizeHigh：标识映射对象的高位大小。
- dwMaximumSizeLow：标识映射对象的低位大小。
- lpName：标识映射对象的名称。
- 返回值：如果函数执行成功，返回值是映射对象的句柄，如果执行失败，返回值为 NULL。

(2) MapViewOfFile 函数。该函数将内存映射对象映射到进程的地址空间中，其语法如下：

```
LPVOID MapViewOfFile(HANDLE hFileMappingObject, DWORD dwDesiredAccess, DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow, DWORD dwNumberOfBytesToMap);
```

参数说明：

- hFileMappingObject：标识映射对象句柄，通常为 CreateFileMapping 函数的返回值。
- dwDesiredAccess：确定访问模式。
- dwFileOffsetHigh：标识映射对象高位的偏移量。
- dwFileOffsetLow：标识映射对象低位的偏移量。
- dwNumberOfBytesToMap：标识映射的范围。
- 返回值：如果函数执行成功，返回值是指向映射对象的起始地址，如果函数执行失败，返回值为 NULL。

## 实现过程

(1) 新建一个基于对话框的应用程序。在对话框中添加按钮控件、文本编辑控件。

(2) 在对话框初始化时创建内存映射对象。

```
m_hShareMem = CreateFileMapping((HANDLE)0xffffffff, NULL,
 PAGE_READWRITE, 0, 10000, "MemFile"); //创建内存映射对象
//将内存映射对象映射到地址空间
m_pViewData = MapViewOfFile(m_hShareMem, FILE_MAP_WRITE, 0, 0, 0);
```

(3) 处理“写入数据”按钮的单击事件，向共享内存中写入数据。

```
void CShareMemDlg::OnWrite()
{
 CString str;
 m_Write.GetWindowText(str); //获得写入数据
 strcpy((char*)m_pViewData, (char*)(LPCTSTR)str); //写入数据
}
```

(4) 处理“读取数据”按钮的单击事件，从共享内存中读取数据。

```
void CShareMemDlg::OnRead()
{
 CString str;
 strcpy((char*)(LPCTSTR)str, (char*)m_pViewData); //读取共享内存数据
 m_Read.SetWindowText(str); //显示数据
}
```

## 举一反三

根据本实例的设计思路和一些技术要点，读者还可以：

- 利用内存映射文件复制磁盘文件。

## 12.4 网上资源共享

在局域网上有许多可以共享的资源，每个人也有许多希望和别人分享的资源。因此资源共享成为一项非常重要的课题。本节将介绍如何在局域网实现资源共享。

## 实例 359 获得网上共享资源

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\12\359

## 实例说明

在有局域网的单位中, 如果能将一些资源共享, 不但能提高资源的有效利用, 还能提高工作效率, 节省资源。但资源共享后, 如果不进行有效的管理和利用, 将不能高效、合理地使用网上资源。本例实现了列举局域网上的共享资源的功能, 可以方便查看局域网上的资源。程序运行结果如图 12.12 所示。

## 技术要点

本实例主要使用了 Windows API 函数库中的 WnetOpenEnum、WnetEnumResource 和 WnetCloseEnum 函数。

## 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“获得网上共享资源”。
- (2) 在窗体上添加一个列表视图控件。
- (3) 主要程序代码。



图 12.12 获得网上共享资源

```
m_imagelist.Create(16,16,ILC_COLOR24|ILC_MASK,0,0);//创建图像列表
//加载图标资源
m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON3));
m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON1));
m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON2));
m_imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON4));
HICON hIcon=::LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDR_MAINFRAME));
m_Tree.SetImageList(&m_imagelist,LVSIL_NORMAL);//关联图像列表
m_Root=m_Tree.InsertItem("整个网络",0,0); //插入根节点
m_Tree.Expand(m_Root,TVE_EXPAND); //展开根节点
//枚举网络资源
DWORD Count=0xFFFFFFFF,Bufsize=4096,Res;
NETRESOURCE* nRes;
NETRESOURCE* nRes1;
NETRESOURCE* nRes2;
NETRESOURCE* nRes3;
HANDLE lphEnum;
LPVOID Buf = new char[4096];
LPVOID Buf1 = new char[4096];
LPVOID Buf2 = new char[4096];
LPVOID Buf3 = new char[4096];
Res = WNetOpenEnum(RESOURCE_GLOBALNET, RESOURCETYPE_ANY, RESOURCEUSAGE_CONTAINER,
NULL,&lphEnum);
Res=WNetEnumResource(lphEnum,&Count,Buf,&Bufsize);
nRes=(NETRESOURCE*)Buf;
//循环枚举下一层资源
for(DWORD n=0;n<Count;n++,nRes++)
{
 DWORD Count1=0xFFFFFFFF;
 Res = WNetOpenEnum(RESOURCE_GLOBALNET, RESOURCETYPE_ANY, RESOURCEUSAGE_CONTAINER,
nRes,&lphEnum);
 Res=WNetEnumResource(lphEnum,&Count1,Buf1,&Bufsize);
 nRes1=(NETRESOURCE*)Buf1;

 for(DWORD i=0;i<Count1;i++,nRes1++)
 {
 m_Group = m_Tree.InsertItem(nRes1->lpRemoteName,1,1,m_Root);
 DWORD Count2=0xFFFFFFFF;
 Res = WNetOpenEnum(RESOURCE_GLOBALNET, RESOURCETYPE_ANY, RESOURCEUSAGE_CONTAINER,nRes1,&lphEnum);
 Res=WNetEnumResource(lphEnum,&Count2,Buf2,&Bufsize);
 nRes2=(NETRESOURCE*)Buf2;
```

```

for(DWORD j=0;j<Count2;j++,nRes2++)
{
 CString sName = nRes2->lpRemoteName;
 sName = sName.Right(sName.GetLength()-2);
 m_Name = m_Tree.InsertItem(sName,2,2,m_Group);
 DWORD Count3=0xFFFFFFFF;
 Res = WNetOpenEnum(RESOURCE_GLOBALNET, RESOURCETYPE_ANY, RESOURCEUSAGE_
CONNECTABLE, nRes2,&lphEnum);
 Res=WNetEnumResource(lphEnum,&Count3,Buf3,&Bufsize);
 nRes3=(NETRESOURCE*)Buf3;
 for(DWORD k=0;k<Count3;k++,nRes3++)
 {
 CString sShare = nRes3->lpRemoteName;
 sShare = sShare.Right(sShare.GetLength()-3-sName.GetLength());
 m_Tree.InsertItem(sShare,3,3,m_Name);
 }
}
}
}
delete Buf;
delete Buf1;
delete Buf2;
delete Buf3;
WNetCloseEnum(lphEnum);

```

### 举一反三

根据本实例，读者可以：

- 获取工作组和相关工作组内的计算机名称。

## 实例 360 映射网络驱动器

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\12\360

### 实例说明

Windows 提供的“映射网络驱动器”命令准许用户在“我的电脑”或“Windows 资源管理器”中显示网络资源，这使得网络资源更易于查找。对于经常使用的网络资源或者当准确知道想要连接的网络路径和资源名时，可以使用“映射网络驱动器”。运行程序，输入驱动器的名称、网络资源路径，单击“连接”按钮，即可实现映射网络驱动器。运行结果如图 12.13 所示。

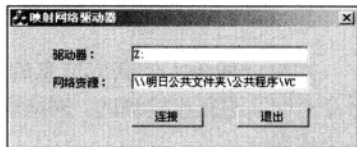


图 12.13 映射网络驱动器

### 技术要点

本实例的实现主要是利用 WNetAddConnection2 函数连接到指定的网络资源，并用指定的盘符代表这个连接。使用 WNetAddConnection2 函数取消到指定网络资源的连接。

WNetAddConnection2 函数：创建同一个网络资源的连接。语法如下：

DWORD WNetAddConnection2( LPNETRESOURCE lpNetResource,LPCTSTR lpPassword,LPCTSTR lpUsername,DWORD dwFlags );

参数说明：

- lpNetResource：在 NETRESOURCE 结构中设置了下述字段，对要连接的网络资源进行了定义，分别为 dwType、lpLocalName（可为 vbNullString 表示用默认提供者）、lpRemoteName、lpProvider。该结构的其他所有变量都会被忽略。
- lpPassword：可选的一个密码。如为 vbNullString，表示采用当前用户的默认密码。如为一个空字符串，则不用任何密码。
- lpUsername：用于连接的用户名。如为 vbNullString，表示使用当前用户。
- dwFlags：设为零；或指定常数 CONNECT\_UPDATE\_PROFILE，表示创建永久性连接。



## 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“映射网络驱动器”。
- (2) 向窗体中添加两个文本编辑控件和两个按钮控件。
- (3) 主要程序代码。

```
void CMappingDlg::OnButjoin()
{
 UpdateData(true);
 NETRESOURCE net;
 DWORD MyErr;
 net.dwScope = RESOURCE_GLOBALNET;
 net.dwType = RESOURCETYPE_DISK;
 net.dwDisplayType = RESOURCEDISPLAYTYPE_SHARE;
 net.dwUsage = RESOURCEUSAGE_CONNECTABLE;
 net.lpLocalName = m_drive.GetBuffer(0); //驱动器
 net.lpRemoteName = m_path.GetBuffer(0); //网络路径
 net.lpComment=NULL;
 net.lpProvider=NULL;
 //创建网络资源连接
 MyErr = WNetAddConnection2(&net, NULL, NULL,CONNECT_UPDATE_PROFILE);
 if(MyErr == NO_ERROR)
 {
 MessageBox("网络驱动器映射成功!", "映射信息提示");
 }
 else
 {
 MessageBox("网络驱动映射器失败!", "映射信息提示");
 }
 m_drive.ReleaseBuffer();
 m_path.ReleaseBuffer();
}
```

## 举一反三

根据本实例，读者可以：

- 映射局域网中的打印机。

## 12.5 套接字应用

网络已成为人们获取信息的主要手段，网络连接是网络传播的基础。本节主要介绍用 Ping 命令对网络进行连接测试、网络语音电话、网络流量监控、获取 Modem 的相应状态和判断 TCP/IP 是否安装。

### 实例 361 网络聊天室

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\12\361

### 实例说明

网络聊天在现在的年轻人中是比较流行的。本实例将介绍如何在局域网中实现网络聊天。在局域网的机器上同时运行局域网聊天的程序的服务器端和客户端。在服务器端单击“监听”按钮，在客户端输入服务器端的服务器名称和昵称，单击“连接”按钮，即可进行连接。此时，在不同的计算机上运行的客户端之间就可以聊天了。本实例实现了一个网络聊天室程序，如图 12.14 所示。

### 技术要点

CSocket 类对 Windows Socket API 进行高层次的封装。它支持同步操作，可以单独使用，

可以与 CSocketFile、CArchive 类一起实现数据的发送和接收。下面介绍 CSocket 的主要方法。

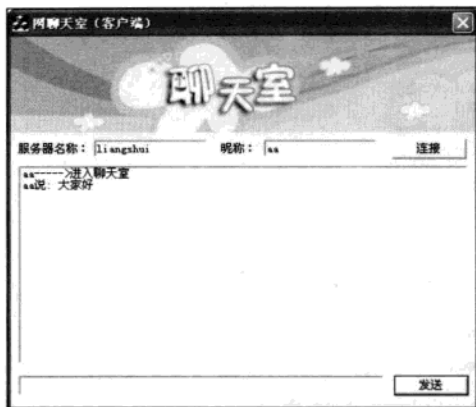


图 12.14 网络聊天室

(1) Create 方法。该方法用于创建一个套接字。语法如下：

```
BOOL Create(UINT nSocketPort = 0, int nSocketType = SOCK_STREAM, LPCTSTR lpszSocketAddress = NULL);
```

参数说明：

- nSockPort：确定套接字端口号。
- nSocketType：确定套接字类型。
- lpszSocketAddress：确定套接字 IP 地址。

(2) Listen 方法。该函数用于监听套接字的连接请求。语法如下：

```
BOOL Listen(int nConnectionBacklog = 5);
```

参数说明：

- nConnectionBacklog：表示等待连接的最大队列长度。

(3) Send 方法。该方法用于发送数据到连接的套接字上。语法如下：

```
virtual int Send(const void* lpBuf, int nBufLen, int nFlags = 0);
```

参数说明：

- lpBuf：标识要发送数据的缓冲区。
- nBufLen：确定缓冲区的大小。
- nFlags：标识函数调用方法。

(4) Receive 方法。该方法用于从一个套接字上接收数据。语法如下：

```
virtual int Receive(void* lpBuf, int nBufLen, int nFlags = 0);
```

参数说明：

- lpBuf：接收数据的缓冲区。
- nBufLen：确定缓冲区的长度。
- nFlags：确定函数的调用模式，可选值如下。
  - MSG\_PEEK：用来查看传来的数据，在序列前端的数据会被复制一份到返回缓冲区中，但是这个数据不会从序列中被移走。
  - MSG\_OOB：用来处理 Out-Of-Band 数据。

## 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“局域网聊天程序（服务器端）”。
- (2) 在窗体上添加一个图片控件和一个按钮控件。
- (3) 从 CSocket 类派生一个新类 CClientSocket，在头文件中引用对话框的头文件和 afxsock.h 头

文件, 并对对话框类进行前导声明。

(4) 绑定套接字, 并开始监听客户端, 代码如下:

```
void CServerDlg::OnOK()
{
 this->UpdateData();
 m_pSocket = new CServerSocket(this);
 if (!m_pSocket->Create(70))
 {
 MessageBox("套接字创建失败");
 delete m_pSocket;
 m_pSocket = NULL;
 return;
 }

 if (!m_pSocket->Listen())
 MessageBox("监听失败");
}
```

(5) 在对话框中添加 AcceptConnect 方法, 用于接受客户端的连接。

```
void CServerDlg::AcceptConnect()
{
 CClientSocket* socket = new CClientSocket(this);
 //接受客户端的连接
 if (m_pSocket->Accept(*socket))
 m_socketlist.AddTail(socket);
 else
 delete socket;
}
```

(6) 在对话框中添加 ReceiveData 方法, 用于接收客户端传来的数据, 代码如下:

```
void CServerDlg::ReceiveData(CClientSocket* socket)
{
 char bufferdata[BUFFERSIZE];

 //接收客户端传来的数据
 int result = socket->Receive(bufferdata, BUFFERSIZE);
 bufferdata[result] = 0;

 POSITION pos = m_socketlist.GetHeadPosition();
 //将数据发送给每个客户端
 while (pos != NULL)
 {
 CClientSocket* socket = (CClientSocket*)m_socketlist.GetNext(pos);
 if (socket != NULL)
 socket->Send(bufferdata, result);
 }
}
```

(7) 新建一个基于对话框的应用程序, 将窗体标题改为“局域网聊天程序 (客户端)”。

(8) 在窗体上添加 1 个图片控件、3 个文本编辑控件、1 个列表框控件和 2 个按钮控件。

(9) 从 CSocket 类中派生一个子类 CMysocket。在头文件中引用 Afxsock.h 头文件, 目的是使用 CSocket 类; 引用主对话框的头文件, 并对主对话框进行前导声明, 因为在 CMysocket 类中需要定义主对话框类指针。

(10) 处理“发送”按钮的单击事件, 发送数据到服务器。代码如下:

```
void CClientDlg::OnButtonsend()
{
 // TODO: Add your control notification handler code here
 CString str, temp;
 m_info.GetWindowText(str);
 if (str.IsEmpty() || m_name.IsEmpty())
 return;
 temp.Format("%s说: %s", m_name, str);
 int num = pMysocket->Send(temp.GetBuffer(
 (temp.GetLength()), temp.GetLength()));
 m_info.SetWindowText("");
 m_info.SetFocus();
}
```

(11) 在主对话框中定义一个 CMysocket 对象指针。添加 ReceiveData 成员方法, 用于接收服务器传来的数据, 代码如下:

```
void CClientDlg::ReceiveData()
{
}
```

```

char buffer[200];
//接收传来的数据
int factdata = pMysocket->Receive(buffer,200);

buffer[factdata] = '\0';
CString str;
str.Format("%s",buffer);
int i = m_list.GetCount();
//将数据添加到列表框中
m_list.InsertString(m_list.GetCount(),str);
}

```

(12) 连接服务器，代码如下：

```

void CClientDlg::OnButtonjoin()
{
 // TODO: Add your control notification handler code here
 UpdateData(true);
 CString servename = m_servename; //读取服务器名称
 int port;
 port = 70; //获取端口

 if (!pMysocket->Connect(servename,port)) //连接服务器
 {
 MessageBox("连接服务器失败!");
 return;
 }
 CString str;
 str.Format("%s----->%s",m_name,"进入聊天室");
 int num = pMysocket->Send(str.GetBuffer(0),str.GetLength());
}

```

### 举一反三

根据本实例，读者可以：

- 利用 Windows Socket API 函数开发一个基于 TCP 的局域网聊天程序。

## 实例 362 语音实时通信

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\12\362

### 实例说明

由于聊天程序的广泛应用，单纯的文字交流已经不能满足广大用户的要求，而网络语音通信则越来越受到广大用户的欢迎。本例制作了一个简单的网络语音通信程序，该程序是一个单向的语音录制与播放的工具。客户端用来与服务器端建立连接，并将录制的声音数据发送到服务器端，而服务器端接收到语音数据后再将其播放出来。如图 12.15 所示。

### 技术要点

本例使用如下 API 函数进行音频采样及播放。

(1) WaveInOpen 函数。WaveInOpen 函数打开录制波形音频设备，语法如下：

```

MMRESULT waveInOpen(LPHWAVEIN phwi,UINT uDeviceID,LPWAVEFORMATEX pwfx,DWORD dwCallback,DWORD
dwCallbackInstance,DWORD fdwOpen);

```

参数说明：

- phwi：波形音频输入设备句柄。
- uDeviceID：波形音频输入设备 ID。
- pwfx：WAVEFORMATEX 结构指针。
- dwCallback：回调函数，处理录音中的消息。
- dwCallbackInstance：回调的用户数据。
- fdwOpen：音频设备打开方式。

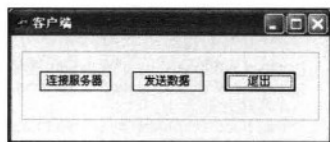


图 12.15 语音实时通信





(2) waveInPrepareHeader 函数。waveInPrepareHeader 函数为录音准备缓冲区, 语法如下。

MMRESULT waveInPrepareHeader( HWAVEIN hwi, LPWAVEHDR pwh, UINT cbwh );

参数说明:

- hwi: 波形音频输入设备句柄。
- pwh: WAVEHDR 结构指针。
- cbwh: WAVEHDR 结构大小。

WaveOutOpen 和 waveOutPrepareHeader 函数的参数与前面的函数基本相同, 所以这里就不再介绍。

## 实现过程

(1) 新建一个基于对话框的工程, 建立服务器端应用程序。

(2) 在对话框中添加编辑框控件, 用于设置监听的端口号, 添加按钮控件, 用于执行服务器端的监听操作。

(3) 在服务器窗口的初始化方法中调用 InitAudio 方法用于对音频输出数据进行初始化操作。

实现代码如下:

```
void CUuuuDlg::InitAudio()
{
 waveform.wFormatTag = WAVE_FORMAT_PCM; // 采样方式, PCM(脉冲编码调制)
 waveform.nChannels = 2; // 双声道
 waveform.nSamplesPerSec = 11025; // 采样率 11.025kHz
 waveform.nAvgBytesPerSec = 11025; // 数据率 11.025KB/s
 waveform.nBlockAlign = 1; // 最小块单元, wBitsPerSample × nChannels / 8
 waveform.wBitsPerSample = 8; // 样本大小为 8bit
 waveform.cbSize = 0;

 lpInWaveHdr[0].dwBufferLength = 4096;
 lpInWaveHdr[0].lpData = lpInBuf;
 lpInWaveHdr[0].dwBytesRecorded = 4096;
 lpInWaveHdr[0].dwFlags = 0; // WHDR_BEGINLOOP | WHDR_ENDLOOP;
 lpInWaveHdr[0].dwLoops = 0;
 lpInWaveHdr[0].dwUser = 0;
 lpInWaveHdr[0].lpNext = NULL;
 lpInWaveHdr[0].reserved = 0;

 lpOutWaveHdr[0].dwBufferLength = 4096;
 lpOutWaveHdr[0].lpData = lpOutBuf;
 lpOutWaveHdr[0].dwBytesRecorded = 4096;
 lpOutWaveHdr[0].dwFlags = 0; // WHDR_BEGINLOOP | WHDR_ENDLOOP;
 lpOutWaveHdr[0].dwLoops = 0;
 lpOutWaveHdr[0].dwUser = 0;
 lpOutWaveHdr[0].lpNext = NULL;
 lpOutWaveHdr[0].reserved = 0;
 // 打开放音设备
 waveOutOpen(&m_hWaveOut, WAVE_MAPPER, &waveform, (DWORD)m_hWnd, 0, CALLBACK_WINDOW);
 waveOutPrepareHeader(m_hWaveOut, lpOutWaveHdr, 4096);
 // 设置音量大小
 waveOutSetVolume(m_hWaveOut, 32765);
}
```

(4) 当服务器接收到音频数据后将其写入缓冲区, 并通过音频输出函数播放出来。实现代码如下:

```
// 接收音频数据
void CUuuuDlg::OnReceiveAudioData(CClientSocket *sock)
{
 HGLOBAL hGlobal = GlobalAlloc(GMEM_MOVEABLE, 9999);
 char* lpBuf = (char*)GlobalLock(hGlobal);

 memset(lpBuf, 0, 9999);
 int size = sock->Receive(lpBuf, 9999);

 memset(lpOutBuf, 0, 4096);
 if (size <= 4096)
 {
 memcpy(lpOutBuf, lpBuf, size);
 PlayAudio();
 }
}
```

```
GlobalUnlock(hGlobal);
GlobalFree(hGlobal);
}
void CUuuuDlg::PlayAudio()
{
 waveOutWrite(m_hWaveOut,lpOutWaveHdr,sizeof(WAVEHDR));
}
```

(5) 新建一个基于对话框的工程，建立客户端应用程序。

(6) 在应用程序窗体上添加 3 个按钮控件，并设置其 Caption 属性值为“连接服务器”、“发送数据”和“退出”。

(7) 在窗体上单击“连接服务器”按钮，实现与服务器端建立连接。实现代码如下：

```
void CKinescodeDlg::OnLinkserver()
{
 LABEL1: CLogin login;
 if (login.DoModal()==IDOK)
 {
 if (m_pAudioSock != NULL)
 delete m_pAudioSock;
 m_pAudioSock = new CClientSocket(this,tpAudio);

 CString port = login.m_Port;
 CString serveraddr = login.m_ServerAddr;

 if (!m_pAudioSock->Create())
 {
 delete m_pAudioSock;
 m_pAudioSock = NULL;
 MessageBox("操作失败");
 return ;
 }
 UINT i_port = atoi(port);
 if ((m_pAudioSock->Connect(serveraddr,i_port)==FALSE))
 {
 if (MessageBox("连接服务器失败，是否尝试重新连接？",
 "提示",MB_YESNO)==IDYES)
 {
 delete m_pAudioSock;
 m_pAudioSock = NULL;
 goto LABEL1; //重新连接
 }
 else
 return;
 }

 m_IsSend = TRUE;
 }
}
```

(8) 在窗体中单击“发送数据”按钮，将从声音输入设备中获取音频数据并发送到服务器端。实现代码如下：

```
void CKinescodeDlg::InitAudio()
{
 waveform.wFormatTag = WAVE_FORMAT_PCM; // 采样方式，PCM(脉冲编码调制)
 waveform.nChannels = 2; // 双声道
 waveform.nSamplesPerSec = 11025; // 采样率11.025kHz
 waveform.nAvgBytesPerSec = 11025; // 数据率11.025KB/s
 waveform.nBlockAlign = 1; // 最小块单元，wBitsPerSample×nChannels/8
 waveform.wBitsPerSample = 8; // 样本大小为8bit
 waveform.cbSize = 0;

 lpInWaveHdr[0].dwBufferLength = 4096;
 lpInWaveHdr[0].lpData = lpInbuf;
 lpInWaveHdr[0].dwBytesRecorded = 0;
 lpInWaveHdr[0].dwFlags = 0;
 lpInWaveHdr[0].dwLoops = 0;
 lpInWaveHdr[0].dwUser = 0;
 lpInWaveHdr[0].lpNext = NULL;
 lpInWaveHdr[0].reserved = 0;

 lpInWaveHdr[1].dwBufferLength = 4096;
 lpInWaveHdr[1].lpData = lpInbuf1;
 lpInWaveHdr[1].dwBytesRecorded = 0;
```



```

lpInWaveHdr[1].dwFlags = 0;
lpInWaveHdr[1].dwLoops = 0;
lpInWaveHdr[1].dwUser = 0;
lpInWaveHdr[1].lpNext = NULL;
lpInWaveHdr[1].reserved = 0;

lpOutWaveHdr[0].dwBufferLength = 4096;
lpOutWaveHdr[0].lpData = lpOutbuf;
lpOutWaveHdr[0].dwBytesRecorded = 4096;
lpOutWaveHdr[0].dwFlags = 0;
lpOutWaveHdr[0].dwLoops = 0;
lpOutWaveHdr[0].dwUser = 0;
lpOutWaveHdr[0].lpNext = NULL;
lpOutWaveHdr[0].reserved = 0;

//打开录音设备和放音设备
MMRESULT result = waveInOpen(&m_hWaveIn,WAVE_MAPPER,&waveform,(DWORD)m_hWnd
,0,CALLBACK_WINDOW);
waveInPrepareHeader(m_hWaveIn,lpInWaveHdr,4096);
StartRecord();
m_pAudioSock->SetSockOpt(SO_SNDBUF,lpOutWaveHdr->lpData,4096);
}

```

(9) 在窗体中实现 MM\_WIM\_DATA 消息映射函数, 该函数将在音频输入设备向缓冲区中写完数据后被调用。实现代码如下:

```

void CKinescodeDlg::EndRecord()
{
 m_pAudioSock->SetSockOpt(SO_SNDBUF,lpOutWaveHdr->lpData,4096);
 if (m_Change)
 m_pAudioSock->Send(lpInWaveHdr[0].lpData,4096);
 else
 m_pAudioSock->Send(lpInWaveHdr[1].lpData,4096);
 m_Change !=m_Change;
 StartRecord();
}

```

### 举一反三

根据本实例, 读者可以:

- 开发网络视频聊天程序。

## 实例 363 视频聊天室

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\12\363

### 实例说明

由于聊天程序的广泛应用, 单纯的文字交流已经不能满足广大用户的要求, 而网络聊天软件则越来越受到广大用户的欢迎。本例制作了一个服务器端利用视频摄像头捕获视频信息发送到客户端并显示视频图像的视频聊天软件, 如图 12.16 所示。

### 技术要点

本例主要使用 VFW 技术实现视频的捕捉。VFW 是 Microsoft 公司推出的一个数字视频软件包, 它能使应用程序数字化并播放从传统模拟视频源得到的视频剪辑。VFW 主要由一组动态库构成, 在安装操作系统时会自动安装。为了支持 VFW, Visual C++ 提供了 vfw32.lib、msacm32.lib、winmm.lib 库文件。在这些库文件中提供了多个函数和宏用于视频应用程序的开发。下面介绍主要的视频函数。

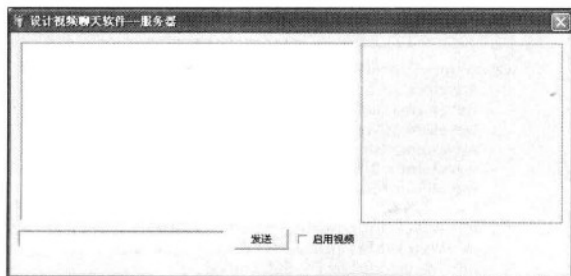


图 12.16 视频聊天室

(1) capCreateCaptureWindow 函数。该函数用于创建一个视频捕捉窗口。语法如下:

```
HWND VFAPAPI capCreateCaptureWindow(LPCSTR lpszWindowName,DWORD dwStyle,int x, int y, int nWidth, int nHeight,HWND hWnd,int nID);
```

参数说明:

- lpszWindowName: 标识窗口的名称。
- dwStyle: 标识窗口风格。
- x、y: 标识窗口的左上角坐标。
- nWidth、nHeight: 标识窗口的宽度和高度。
- hWnd: 标识父窗口句柄。
- nID: 标识窗口 ID。
- 返回值: 视频捕捉窗口句柄。

(2) capDriverConnect 宏。该宏主要将视频捕捉窗口连接到视频驱动程序上。语法如下:

```
BOOL capDriverConnect(hwnd, iIndex);
```

参数说明:

- hwnd: 标识视频捕捉窗口句柄。
- iIndex: 标识驱动程序, 范围是 0~9。

(3) capPreviewRate 宏。该宏在预览模式下设置帧的显速率。语法如下:

```
BOOL capPreviewRate(hwnd, wMS);
```

参数说明:

- hwnd: 标识视频捕捉窗口句柄。
- wMS: 标识速率, 单位是毫秒。

(4) capPreview 宏。该宏用于激活或禁止预览模式。在预览模式下, 视频设备捕捉的数据以帧的形式传递到系统内存中, 然后通过 GDI 函数显示在视频捕捉窗口中。语法如下:

```
BOOL capPreview(hwnd, f);
```

参数说明:

- hwnd: 标识视频捕捉窗口句柄。
- f: 标识激活或禁止预览模式。

## 实现过程

(1) 新建一个基于对话框的工程, 建立服务器端应用程序。

(2) 在对话框中添加编辑框控件, 用于发送文本信息, 添加格式文本编辑控件, 用于显示发送和接收的文本信息。

(3) 在窗体初始化时创建 CSocket 类并初始化视频头, 将通过视频头获取的图像显示在窗体中。实现代码如下:

```
BOOL CServerDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//系统代码省略
 m_ServerSock.Create(845);
 m_ServerSock.Listen();
 //创建视频捕获窗口并返回句柄
 m_hWndVideo = capCreateCaptureWindow(NULL,WS_POPUP,
 1, 1, 10, 10, m_hWnd, 0);
 if (capDriverConnect(m_hWndVideo, 0))
 {
 ::SetParent(m_hWndVideo,*this);//将视频捕获窗口放在窗体上
 //设置捕获窗口样式
 ::SetWindowLong(m_hWndVideo,GWL_STYLE,WS_CHILD);
 CRect wndRC;
 m_Panel.GetClientRect(wndRC);
 m_Panel.MapWindowPoints(this,wndRC);
 wndRC.DeflateRect(1,1,1,1);//获取捕获窗口大小
 ::SetWindowPos(m_hWndVideo,NULL,wndRC.left,
 wndRC.top,wndRC.Width(),wndRC.Height(),SWP_NOZORDER);
 }
}
```



```

::ShowWindow(m_hWndVideo, SW_SHOW); //设置捕获窗口位置并显示
CAPDRIVERCAPS caps; //获取捕获窗口信息
capDriverGetCaps(m_hWndVideo, sizeof(caps), &caps);
if (caps.fHasOverlay)
 capOverlay(m_hWndVideo, TRUE);
CAPTUREPARMS params; //获取捕获窗口参数
capCaptureGetSetup(m_hWndVideo, ¶ms, sizeof(params));
params.fYield = TRUE;
params.fAbortLeftMouse = FALSE;
params.fAbortRightMouse = FALSE;
params.fLimitEnabled = FALSE;
params.vKeyAbort = FALSE;
params.fCaptureAudio = FALSE;
//设置捕获窗口参数
capCaptureSetSetup(m_hWndVideo, ¶ms, sizeof(params));
//设置视频回调函数
capSetCallbackOnVideoStream(m_hWndVideo, EncodeCallback);
capPreviewRate(m_hWndVideo, 30); //设置预览比率
capPreview(m_hWndVideo, TRUE); //启用预览
capCaptureSequenceNoFile(m_hWndVideo);
m_bSendImage = FALSE;
}
SetTimer(1, 200, NULL);
return TRUE;
}

```

(4) 实现 EncodeCallback 函数，该函数是视频捕获时使用的回调函数。该函数的实现用于获取视频捕获窗口中的图像数据并存储到 CPackage 类对象中所对应的数据缓冲区中。实现代码如下：

```

LRESULT WINAPI EncodeCallback(HWND hWnd, LPVIDEOHDR lpVHdr)
{
 if (lpVHdr->dwFlags & VHDR_DONE)
 {
 static BOOL bSend = TRUE;
 CServerDlg* pDlg = (CServerDlg*)AfxGetMainWnd();
 int nState = pDlg->m_Video.GetCheck();
 if (nState == 1)
 {
 if (pDlg->m_bSendImage == FALSE)
 {
 //获取图像数据
 BITMAPINFO bmpInfo;
 capGetVideoFormat(pDlg->m_hWndVideo, &bmpInfo,
 sizeof(BITMAPINFO));
 //确定图像数据大小
 int nSize = bmpInfo.bmiHeader.biSizeImage;
 bSend = FALSE;
 HGLOBAL hGlobal = GlobalAlloc(GHND, nSize +
 sizeof(BITMAPINFOHEADER));
 BYTE* pData = (BYTE*)GlobalLock(hGlobal);
 memcpy(pData, &bmpInfo.bmiHeader,
 sizeof(BITMAPINFOHEADER));
 BYTE* pTmp = pData;
 pTmp += sizeof(BITMAPINFOHEADER);
 memcpy(pTmp, lpVHdr->lpData, nSize);
 int nPackSize = sizeof(CPackage) + nSize +
 sizeof(BITMAPINFOHEADER);

 if (pDlg->m_hGlobal != NULL)
 {
 GlobalFree(pDlg->m_hGlobal);
 pDlg->m_hGlobal = NULL;
 }
 pDlg->m_hGlobal = GlobalAlloc(GHND, nPackSize);
 BYTE* pSendData = (BYTE*)GlobalLock(pDlg->m_hGlobal);
 CPackage* pPackage = (CPackage*)pSendData;
 pPackage->m_Type = ptImage;
 pPackage->m_dwContent = nSize + sizeof(BITMAPINFOHEADER);
 pPackage->m_dwData = nSize + sizeof(BITMAPINFOHEADER);
 pPackage->m_dwSize = sizeof(CPackage);
 pTmp = pSendData;
 pTmp += sizeof(CPackage);
 memcpy(pTmp, pData, nSize + sizeof(BITMAPINFOHEADER));
 GlobalUnlock(hGlobal);
 GlobalFree(hGlobal);
 }
 }
 }
}

```

```
GlobalUnlock(pDlg->m_hGlobal);
pDlg->m_bSendImage = TRUE;
}
}
return 1;
}
```

(5) 在窗体上单击“发送”按钮会将文本信息发送到客户端。实现代码如下：

```
void CServerDlg::OnSend()
{
 CString szSendInfo;
 m_SendInfo.GetWindowText(szSendInfo);
 if (!szSendInfo.IsEmpty())
 {
 //填充数据包
 int nLen = szSendInfo.GetLength();
 HGLOBAL hGlobal = GlobalAlloc(GHND, sizeof(CPackage) + nLen);
 BYTE* pData = (BYTE*)GlobalLock(hGlobal);
 CPackage* pPackage = (CPackage*)pData;
 pPackage->m_Type = ptText;
 pPackage->m_dwContent = nLen;
 pPackage->m_dwSize = sizeof(CPackage);
 pPackage->m_dwData = nLen;
 BYTE* pTmp = pData;
 pTmp += sizeof(CPackage);
 memcpy(pTmp, szSendInfo, nLen);
 m_ClientSock.Send(pData, sizeof(CPackage) + nLen);
 GlobalUnlock(hGlobal);
 GlobalFree(hGlobal);
 m_SendInfo.SetWindowText("");
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel("服务器say: \n");
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel(szSendInfo);
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel("\n");
 }
}
```

(6) 在窗体中实现 ReceiveData 方法，该方法用于接收客户端发送到服务器端的文本数据并显示在格式文本编辑控件中。实现代码如下：

```
//接收数据
void CServerDlg::ReceiveData()
{
 static int nSize = sizeof(CPackage) + 1024;
 HGLOBAL hGlobal = GlobalAlloc(GHND, nSize);
 BYTE* pBuffer = (BYTE*)GlobalLock(hGlobal);
 int nFact = m_ClientSock.Receive(pBuffer, nSize); //接收数据
 if (nFact > 0)
 {
 CPackage* pPackage = (CPackage*)pBuffer;
 int nDataLen = pPackage->m_dwContent;
 //显示接收数据
 if (pPackage->m_Type == ptText)
 {
 char* pszData = (char*)pBuffer;
 pszData += sizeof(CPackage);
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel("客户端say: \n");
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel(pszData);
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel("\n");
 }
 }
 GlobalUnlock(hGlobal);
 GlobalFree(hGlobal);
}
```

(7) 在窗体上实现 OnTimer 消息事件，该事件用于发送视频头捕获的图像信息到客户端。实现代码如下：

```
void CServerDlg::OnTimer(UINT nIDEvent)
{
 if (m_bSendImage)
 {
 if (m_hGlobal != NULL)
```



```

 {
 int nSize = GlobalSize(m_hGlobal);
 BYTE* pData = (BYTE*)GlobalLock(m_hGlobal);
 m_ClientSock.Send(pData, nSize); //发送数据
 m_bSendImage = FALSE;
 GlobalUnlock(m_hGlobal);
 }
}
CDialog::OnTimer(nIDEvent);

```

(8) 新建一个基于对话框的应用程序，创建视频聊天程序的客户端。

(9) 在窗体上添加一个编辑框控件，用于记录需要发送的文本信息；在窗体上添加一个格式文本编辑框控件，用于显示发送和接收的文本信息；添加一个按钮控件，用于向服务器端发送文本信息；添加一个图片控件，用于显示服务器端传递过来的视频图像信息。

(10) 单击窗体上的“发送”按钮，将编辑框控件中的文本信息发送到服务器端。实现代码如下：

```

void CClientDlg::OnSend()
{
 CString szSendInfo;
 m_SendContent.GetWindowText(szSendInfo);
 if (!szSendInfo.IsEmpty())
 {
 //填充数据包
 int nLen = szSendInfo.GetLength();
 HGLOBAL hGlobal = GlobalAlloc(GHND, sizeof(CPackage) + nLen);
 BYTE* pData = (BYTE*)GlobalLock(hGlobal);
 CPackage *pPackage = (CPackage*) pData;
 pPackage->m_Type = ptText;
 pPackage->m_dwContent = nLen;
 pPackage->m_dwSize = sizeof(CPackage);
 pPackage->m_dwData = nLen;
 BYTE* pTmp = pData;
 pTmp += sizeof(CPackage);
 memcpy(pTmp, szSendInfo, nLen);
 m_ClientSock.Send(pData, sizeof(CPackage) + nLen);
 GlobalUnlock(hGlobal);
 GlobalFree(hGlobal);
 m_SendContent.SetWindowText("");
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel("客户端say: \n");
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel(szSendInfo);
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel("\n");
 }
}

```

(11) 在窗体类中实现 HandleRecData 方法，该方法用于接收服务器端向客户端发送的文本数据和视频图像数据。实现代码如下：

```

//处理接收的数据
void CClientDlg::HandleRecData(CPackage *pPackage)
{
 if (pPackage != NULL)
 {
 if (pPackage->m_Type == ptText) //文本数据
 {
 char* szData = (char*)pPackage->m_Data;
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel("服务器say: \n");
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel(szData);
 m_InfoList.SetSel(-1, 0);
 m_InfoList.ReplaceSel("\n"); //添加换行符
 }
 else if (pPackage->m_Type == ptImage) //图像数据
 {
 BITMAPINFOHEADER bmpHeader;
 BITMAPINFO bmpInfo;
 bmpInfo.bmiHeader = bmpHeader;
 memcpy(&bmpHeader, pPackage->m_Data, sizeof(bmpHeader));
 BYTE* pBmpData = pPackage->m_Data;

```

```

pBmpData += sizeof(bmpHeader);
CDC* pDC = GetDC();
HBITMAP hBmp = CreateDIBitmap(pDC->m_hDC, &bmpHeader,
 CBM_INIT, pBmpData,
 (BITMAPINFO*)&bmpHeader, DIB_RGB_COLORS);
HBITMAP hOldBmp = m_Image.SetBitmap(hBmp);
if (hOldBmp != NULL)
{
 DeleteObject(hOldBmp);
}
//显示图像
//...
ReleaseDC(pDC);
}
}
}

```

### 举一反三

根据本实例，读者可以：

- 录制视频过程。

## 12.6 其 他

下面再介绍几个有关局域网的其他相关实例，以帮助读者更好地掌握局域网方面的知识。

### 实例 364 获得拨号网络的列表

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\12\364

#### 实例说明

为了方便拨号上网，可以将拨号网络列表添加在自己的程序内。运行程序，列表框将显示拨号网络列表，选择列表中的拨号链接，即可拨号上网，并随时显示当前网络连接状态。程序运行结果如图 12.17 所示。

#### 技术要点

本例用 RasEnumEntries 函数，将本机内所有“我的连接”放置在列表中。

RasEnumEntries 函数获取一个项的设置值，语法如下：

```

DWORD RasEnumEntries (LPCTSTR reserved, LPTCSTR lpszPhonebook, LPRASENTRYNAME lprasentryname, LPDWORD
lpcb, LPDWORD lpcEntries);

```

参数说明：

- reserved：保留变量，必须为 NULL。
- lpszPhonebook：拨号连接全路径。
- lprasentryname：RASENTRYNAME 结构，用于获得拨号连接的缓冲区。
- lpcb：缓冲区的大小。
- lpcEntries：返回拨号连接的数量。

#### 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“获得拨号网络的列表”。
- (2) 向窗体中添加一个列表框控件。

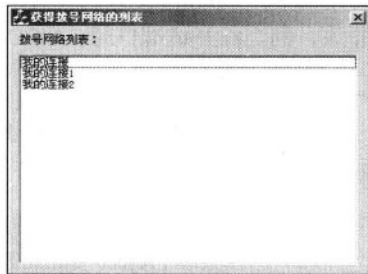


图 12.17 获得拨号网络的列表



## (3) 主要程序代码。

```
DWORD ent,i,j;
RASENTRYNAME buf;
buf.dwSize=264;
j = 256 * buf.dwSize;
RasEnumEntries(NULL,NULL,&buf,&j,&ent); //枚举连接列表
for(i=0;i<ent;i++)
{
 m_List.InsertString(i,buf.szEntryName); //插入连接列表
}
```

## 举一反三

根据本实例，读者可以：

- 获得局域网络的资源情况。

## 实例 365 获取计算机上串口的数量

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\12\365

## 实例说明

在进行串口通信时，需要知道计算机上的串口信息。本例实现了获取计算机中串口数量的功能。运行程序，单击“获取”按钮，程序窗体上将显示当前计算机上串口的数量。实例运行结果如图 12.18 所示。

## 技术要点

本例通过 MSComm 控件的 SetPortOpen 方法来设置并返回通信端口的状态。在使用 SetPortOpen 方法打开端口之前，需要使用 SetCommPort 方法设置一个合法的端口号。如果使用 SetCommPort 方法设置的端口号非法，则当打开该端口时，MSComm 控件将产生一个“设备无效”的错误。

SetPortOpen 方法中语法如下：

```
void SetPortOpen(BOOL bNewValue);
```

参数：bNewValue：为 TRUE 时打开端口，FALSE 时关闭端口。

## 实现过程

(1) 新建一个基于对话框的应用程序，将窗体标题改为“获取计算机上串口的数量”。

(2) 选择菜单 Project/Add To Project，然后选择 Components and Controls...，弹出 Components and Controls Gallery 窗口，双击窗口中的 Registered ActiveX Controls 文件夹，找到 Microsoft Communications Control version6.0 选项，双击它，取默认值，添加控件，单击“Close”按钮，MSComm 控件就添加到控件面板中了。

(3) 在窗体上添加一个 MSComm 控件、一个文本编辑控件和一个按钮控件。

(4) 主要程序代码。

```
void CSerialInterfaceDlg::OnButobtain()
{
 int j=0,i;
 try
 {
 for(i=1;i<6;i++)
 {
 m_msc.SetCommPort(i); //设置端口
 m_msc.SetPortOpen(true); //打开端口
 m_msc.SetPortOpen(false);
 j = j + 1;
 }
 }
}
```

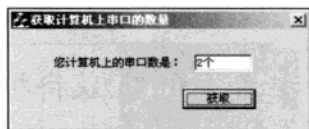


图 12.18 获取计算机上串口的数量

```

catch(...)
{
 CString str;
 str.Format("%d个",j);
 m_Edit.SetWindowText(str);
}
}

```

### 举一反三

根据本实例，读者可以：

- 控制远程计算机休眠；
- 控制远程计算机的鼠标按下。

## 实例 366 检测系统中安装的协议

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\12\366

### 实例说明

本例实现了检测系统中安装的协议的功能。运行程序，单击“检测”按钮，系统中安装的协议将显示在列表中，如图 12.19 所示。


### 技术要点

本例使用 WSAEnumProtocols 函数实现枚举协议的功能，WSAEnumProtocols 函数可以返回当前系统中安装的协议的详细信息，语法如下：

```
int WSAEnumProtocols(LPINT lpiProtocols,LPWSAPROTOCOL_INFO lpProtocolBuffer,LPDWORD lpdwBufferLength);
```

参数说明：

- lpiProtocols：可选参数，为 NULL 时，返回所有可用的协议。
- lpProtocolBuffer：指向枚举结果存放的缓冲区地址。
- lpdwBufferLength：指向枚举结果存储缓冲区大小的变量地址。
- 返回值：枚举协议的个数。

 注意：在使用这些函数之前，需要初始化向程序中导入 ws2\_32.lib 库和头文件 winsock2.h。

### 实现过程

- (1) 新建一个基于对话框的应用程序，将窗体标题改为“检测系统中安装的协议”。
- (2) 向窗体中添加一个列表视图控件和一个按钮控件。
- (3) 主要程序代码。

```

void CProtocolDlg::OnButenumerate()
{
 LPBYTE pBuf; //保存网络协议信息的缓冲区
 DWORD dwLen; //缓冲区的长度
 LPWSAPROTOCOL_INFO pInfo;
 //通过调用WSAEnumProtocols以获得所需缓冲区的大小
 int nRet = WSAEnumProtocols(NULL,NULL,&dwLen);
 if(nRet == SOCKET_ERROR)
 {
 if(WSAGetLastError() != WSAENOBUFS)
 {
 MessageBox("调用失败");
 return;
 }
 }
}

```

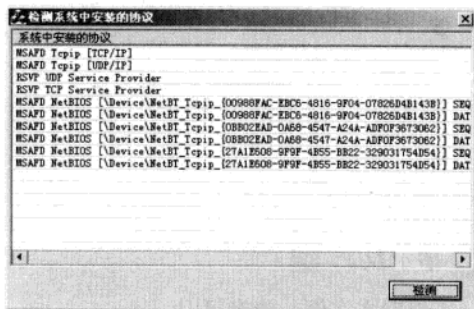


图 12.19 检测系统中安装的协议

```

//检查缓冲区的大小是否可以容纳信息
if(dwLen < sizeof(WSAPROTOCOL_INFO))
{
 MessageBox("缓冲区出现错误");
 return;
}
dwLen++;
pBuf = (LPBYTE)malloc(dwLen); //申请所需的内存
if(pBuf == NULL)
{
 MessageBox("内存分配失败");
 return;
}
//进行枚举, nRet是返回的协议个数
nRet = WSAEnumProtocols(NULL, (LPWSAPROTOCOL_INFO)pBuf, &dwLen);
if(nRet == SOCKET_ERROR)
{
 free(pBuf);
 MessageBox("枚举失败");
 return;
}
//遍历各协议的信息
pInfo = (LPWSAPROTOCOL_INFO)pBuf;
for(int nCount=0; nCount<nRet; nCount++)
{
 //将协议信息添加到列表中
 int i = m_Grid.GetCountPerPage();
 m_Grid.InsertItem(i, pInfo->szProtocol);
 pInfo++;
}
//释放内存
free(pBuf);
}

```

### 举一反三

根据本实例, 读者可以:

- 获取安装协议的详细信息。

## 实例 367 域名解析

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\12\367

### 实例说明

开发网络方面的应用程序时经常使用套接字, 而套接字是针对 TCP/IP 的, 所以针对给出域名的网络地址应先经过一次转换, 转换成 IP 地址后, 才可以继续开发。本实例就是完成将域名解析成地址。程序运行如图 12.20 所示。

### 技术要点

本实例主要通过 gethostbyname 函数实现, gethostbyname 主要是用来获取主机名的函数, 在 Internet 中也可以应用它对域名进行解析。

### 实现过程

- (1) 新建名为 DomainName 的对话框 MFC 工程。
- (2) 在对话框上添加两个文本编辑控件, 设置 ID 属性分别为 IDC\_EDNAME 和 IDC\_EDIP; 添加一个按钮控件, 设置 ID 属性为 IDC\_BTGET, 设置 Caption 属性为“获取”。
- (3) 在工程中添加对 wsck32.lib 库的引用。

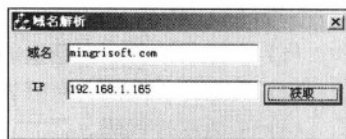


图 12.20 域名解析

(4) 在 StdAfx.h 文件中加入下面语句:

```
#include <afxsock.h>
```

(5) 在 OnInitDialog 中完成类库的初始化, 代码如下:

```
BOOL CDomainNameDlg::OnInitDialog()
{
 ...//此处代码省略
 AfxSocketInit();
 return TRUE;
}
```

(6) 按钮“获取”的实现函数, 该函数通过域名来获得 IP 地址, 代码如下:

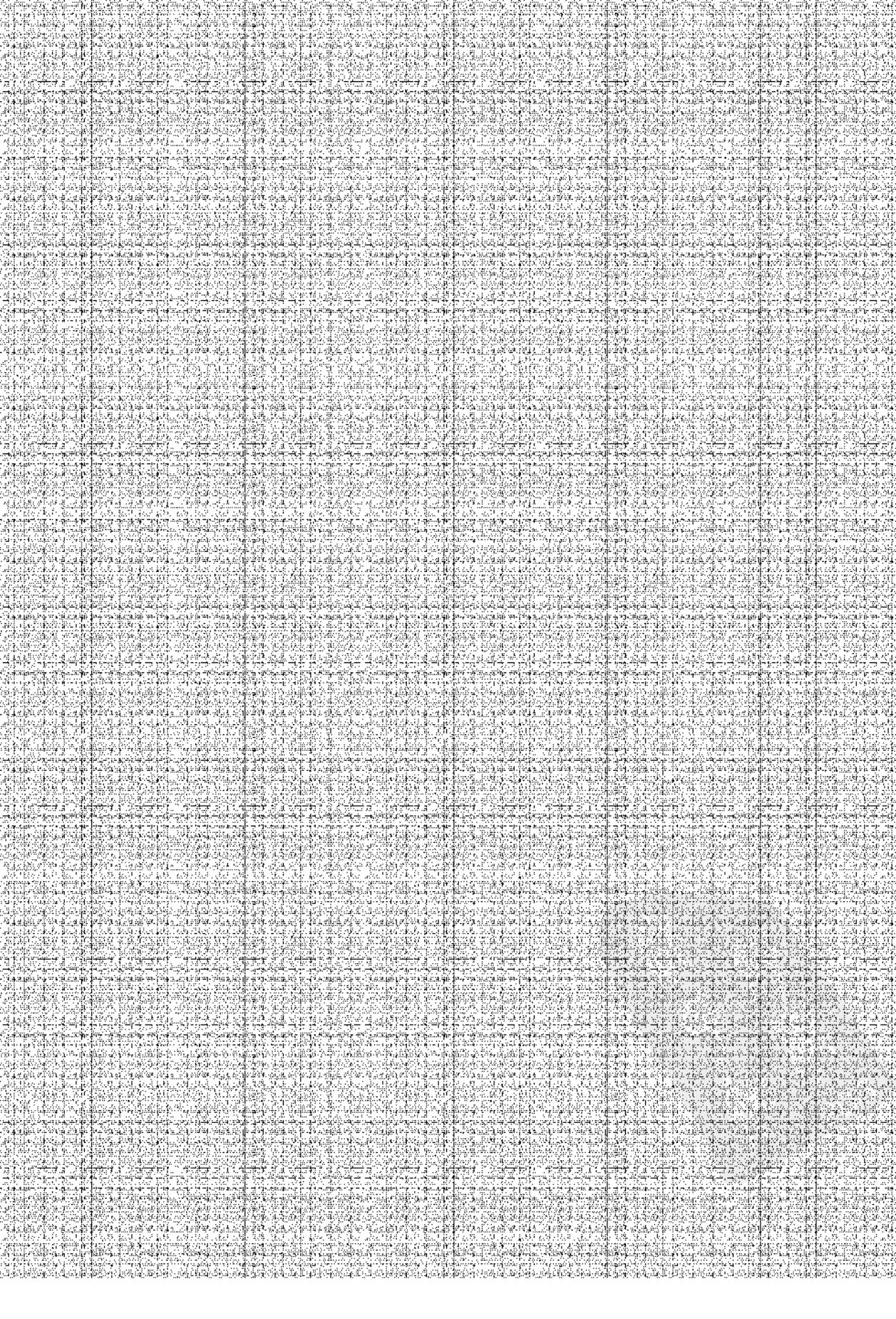
```
void CDomainNameDlg::OnGet()
{
 HOSTENT* hst=NULL;
 CString strname,strip,tmp;
 GetDlgItem(IDC_EDNAME)->GetWindowText(strname);
 struct in_addr ia;
 hst=gethostbyname((LPCTSTR)strname); //获得IP地址
 //格式化IP地址
 for(int i=0;hst->h_addr_list[i];i++){
 memcpy(&ia.s_addr,hst->h_addr_list[i],sizeof(ia.s_addr));
 tmp.Format("%s\\n",inet_ntoa(ia));
 strip+=tmp;
 }
 GetDlgItem(IDC_EDIP)->SetWindowText(strip);
}
```

### 举一反三

根据本实例, 读者可以开发:

- 通过 IP 地址获得域名或主机名。







## 第 13 章

# Web 编程



- 上网控制
- 文件上传与下载
- 邮件管理
- 上网监控
- 浏览器应用
- 网上信息提取
- 其他

Visual C++

## 13.1 上网控制

本节中包括定时登录 Internet 和根据网络连接控制 IE 浏览器启动两个实例,通过这两个实例向读者介绍在应用程序当中控制网络连接方面的知识。

### 实例 368 定时登录 Internet

本实例可以提高工作效率

实例位置: 光盘\mingrisoft\13\368

#### 实例说明

本实例完成在预先设定的时间内登录到设定的网站,运行程序,在“登录网址”文本框中输入所要登录的网站地址,然后通过两个组合框来设置登录时间,单击“确定”按钮后,程序会在系统托盘中运行,设置的时间一到就会登录到网站。实例运行结果如图 13.1 所示。

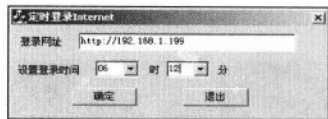


图 13.1 定时登录 Internet

#### 技术要点

本实例主要通过添加在程序中添加定时器来实现定时功能,然后通过 ShellExecute 函数打开 IE 浏览器登录网站。定时器的添加主要是在类向导中添加 WM\_TIMER 消息的处理函数,然后通过 SetTimer 使定时器开始工作。使用 ShellExecute 函数时,将第 3 个参数设为网站地址后可以直接打开 IE 浏览器登录到网站。

#### 实现过程

(1) 新建名为 InternetSetTime 的对话框 MFC 工程。

(2) 在对话框上添加两个静态文本控件,添加文本编辑控件,设置 ID 属性为 IDC\_EDADDRESS,添加两个组合框控件,设置 ID 属性分别为 IDC\_CMBHOUR 和 IDC\_CMBMINU,添加成员变量 m\_hour 和 m\_minu,添加两个按钮控件,设置 ID 属性分别为 IDC\_BTENTER 和 IDC\_BTEXIT。

(3) 按钮“确定”的实现函数,该函数完成定时器的启动,代码如下:

```
void CInternetSetTimeDlg::OnEnter()
{
 GetDlgItem(IDC_EDADDRESS)->GetWindowText(straddress); //获得控件中数据
 CString hour,minu;
 m_hour.GetWindowText(hour);
 m_minu.GetWindowText(minu);
 strtime.Format("%s:%s",hour,minu);
 CString temp;temp.Format("你设置的时间是%s",strtime); //格式化时间字符串
 AfxMessageBox(temp);
 SetTimer(1,1000,NULL); //设置定时器
}
```

(4) 设置定时器完成定时登录,代码如下:

```
void CInternetSetTimeDlg::OnTimer(UINT nIDEvent)
{
 CTime tt;
 tt=CTime::GetCurrentTime(); //获得当前时间
 CString tmp=tt.Format("%H:%M"); //格式化时间字符串
 if(!tmp.CompareNoCase(strtime))
 {
 ::ShellExecute(this->GetSafeHwnd(),"open",straddress,NULL,NULL,SW_SHOW);
 KillTimer(1);
 }
 CDialog::OnTimer(nIDEvent);
}
```



## 举一反三

根据本实例，读者可以开发：

- 定时开机；
- 工作定时提醒。

## 实例 369

根据网络连接控制 IE  
启动

本实例可以提高工作效率

实例位置：光盘\mingrisoft\13\369

## 实例说明

本实例完成定时查看网络连接的功能，如系统中有拨号连接已经连接到 INTERNET，就打开 IE 浏览器，登录到设定的网站。运行程序，在文本框中输入打开 IE 浏览器后登录的网址，程序运行结果如图 13.2 所示。

## 技术要点

本实例通过 RASEnumConnections 函数来枚举系统中的拨号连接，语法如下：

```
DWORD RasEnumConnections(LPRASCONN lprascnn,LPDWORD lpcb,LPDWORD lpcConnections);
```

参数说明：

- lprascnn：RASCONN 结构对象数组指针，存储连接句柄。
- lpcb：设定 lprascnn 参数的大小。
- lpcConnections：连接的数量。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框上添加一个静态文本控件，添加文本编辑控件，设置 ID 属性为 IDC\_EDADDRESS，添加两个按钮控件，设置 ID 属性分别为 IDC\_BTSET 和 IDC\_BTEXIT。
- (3) 按钮“设置”的实现函数，该函数完成定时器的启动，代码如下：

```
void CStartIEDlg::OnSet()
{
 GetDlgItem(IDC_EDADDRESS)->GetWindowText(strcmd); //获得控件中数据
 SetTimer(1,1000,NULL); //设置定时器
}
```

- (4) 设置定时器监控网络连接，代码如下：

```
void CStartIEDlg::OnTimer(UINT nIDEvent)
{
 LPRASCONN lpRasConn = NULL;
 RASCONNSTATUS rasStatus;
 LPHRASCONN g_lpRasConn = NULL;
 DWORD cbBuf=0,cConn=0;
 cbBuf=sizeof(RASCONN);
 lpRasConn=(LPRASCONN)malloc((UINT)cbBuf);
 lpRasConn->dwSize=sizeof(RASCONN);
 ::RasEnumConnections(lpRasConn,&cbBuf,&cConn); // 枚举系统中的拨号连接
 if(cConn>0)
 {
 ::ShellExecute(this->GetSafeHwnd(),"open",strcmd,NULL,NULL,SW_SHOW);
 KillTimer(1);
 }
 CDialog::OnTimer(nIDEvent);
}
```

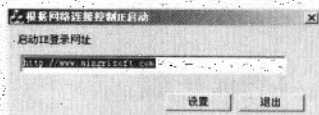


图 13.2 根据网络连接控制 IE 启动



## 举一反三

根据本实例，读者可以：

- 实现定时关闭拨号连接。

## 13.2 文件上传与下载

文件上传与下载是世界上最主要且使用最广泛的 Internet 服务，用户只要能上网，就可以进行文件的上传和下载。下面通过两个实例介绍文件上传与下载程序设计的方法。

## 实例 370 FTP 文件上传程序

本实例可以提高工作效率

实例位置：光盘\mingrisoft\13\370

## 实例说明

文件上传是使用 INTERNET 的用户常用的程序，通过文件上传可以将文件传输到 FTP 服务器上，通过 FTP 服务器实现文件的共享。运行程序，在“IP”和“端口”文本框中输入 FTP 服务器的 IP 地址和端口，如果 FTP 服务器需要验证就输入用户名和密码，如果不需要验证就使用匿名登录，选中“匿名登录”复选框，文本输入完成后单击“连接”按钮，程序将会连接 FTP 服务器，如果连接成功会在列表控件中显示服务器中共享的文件，上传文件只需单击“上传”按钮，选择上传的文件即可。程序运行结果如图 13.3 所示。

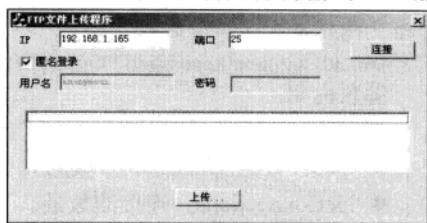


图 13.3 FTP 文件上传程序

## 技术要点

本实例主要通过 CInternetSession 类和 CFtpConnection 类来实现文件的上传，CInternetSession 类是建立一个 INTERNET 连接会话，通过 CInternetSession 类的构造函数构造一个会话，构造函数的语法如下：

```
CInternetSession(LPCTSTR pstrAgent = NULL, DWORD dwContext = 1, DWORD dwAccessType = INTERNET_OPEN_TYPE_PRECONFIG, LPCTSTR pstrProxyName = NULL, LPCTSTR pstrProxyBypass = NULL, DWORD dwFlags = 0);
```

参数说明：

- pstrAgent：能够连接到 INTERNET 的实体。
- dwContext：操作的环境标识，默认是 1。
- dwAccessType：访问的类型，取值如下。
  - INTERNET\_OPEN\_TYPE\_PRECONFIG：通过注册表配置访问连接。
  - INTERNET\_OPEN\_TYPE\_DIRECT：间接的访问连接。
  - NTERNET\_OPEN\_TYPE\_PROXY：通过代理服务器访问连接。
- pstrProxyName：代理服务器名称。
- pstrProxyBypass：代理服务器的验证。
- dwFlags：设置缓存和异步相关信息，默认值是 0。

构造会话完以后就需要获得一个 FTP 的连接，通过 CInternetSession 类的 GetFtpConnection 方法实现，语法如下：

```
CFtpConnection* GetFtpConnection(LPCTSTR pstrServer, LPCTSTR pstrUserName = NULL, LPCTSTR pstrPassword =
NULL, INTERNET_PORT nPort = INTERNET_INVALID_PORT_NUMBER, BOOL bPassive = FALSE);
```

参数说明:

- pstrServer: FTP 的地址。
- pstrUserName: 登录用户名。
- pstrPassword: 登录密码。
- nPort: 登录端口。
- bPassive: 指定被动和主动模式。

通过 GetFtpConnection 方法可以获得 CFtpConnection 类的对象指针, 通过该对象的 PutFile 方法可以进行文件的上传。

如果要获得 FTP 服务器上的列表还需要使用 CFtpFileFind 类, 将 CFtpConnection 类的对象指针传给 CFtpFileFind 类的构造函数, 就可以使用 CFtpFileFind 类的 FindFile 方法进行查找。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加 4 个静态文本控件, 设置 Caption 属性分别为“IP”、“端口”、“用户名”、“密码”; 添加 4 个文本编辑控件, 设置 ID 属性分别为 IDC\_EDIP、IDC\_EDPORT、IDC\_EDUSR 和 IDC\_EDPWD; 添加 1 个复选框控件, 设置 ID 属性为 IDC\_NONAME、Caption 属性为“匿名登录”, 并添加成员变量 m\_noname; 添加 1 个列表框控件, 设置 ID 属性为 IDC\_FTPFILELST, 并添加成员变量 m\_ftplist; 添加两个按钮, 设置 ID 属性分别为 IDC\_BTCONN 和 IDC\_BTUPLOAD, 设置 Caption 属性为“连接”和“上传”。

(3) 添加“连接”按钮的实现函数 OnConnect; 添加“上传”按钮的实现函数 OnUpload。

(4) 在 FtpUploadDlg.h 文件中添加变量声明:

```
CString strusr;
CString strpwd;
CString strip;
CString strport;
BOOL bconnect;
CInternetSession *pInternetSession;
CFtpConnection *pFtpConnection;
```

(5) 在 OnInitDialog 中完成对变量的初始化, 代码如下:

```
BOOL CFtpUploadDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 bconnect=FALSE;
 return TRUE;
}
```

(6) 按钮“连接”的实现函数, 该函数实现连接 FTP 服务器, 代码如下:

```
void CFtpUploadDlg::OnConnect()
{
 //获得控件中数据
 GetDlgItem(IDC_EDIP)->GetWindowText(strip);
 GetDlgItem(IDC_EDPORT)->GetWindowText(strport);
 GetDlgItem(IDC_EDUSR)->GetWindowText(strusr);
 GetDlgItem(IDC_EDPWD)->GetWindowText(strpwd);
 //判断数据是否为空
 if(strip.IsEmpty())
 return;
 if(strport.IsEmpty())
 return;
 if(strusr.IsEmpty())
 return;
 pInternetSession = new CInternetSession("MR", INTERNET_OPEN_TYPE_PRECONFIG); //构造一个会话
 try {
 pFtpConnection = pInternetSession->GetFtpConnection(strip,
 strusr, strpwd, atoi(strport)); //连接FTP
 }
```

```
bconnect=TRUE;
} catch(CInternetException* pEx)
{
 TCHAR szErr[1024];
 pEx->GetErrorMessage(szErr, 1024);
 AfxMessageBox(szErr);
 pEx->Delete();
}
m_ftpfilelst.ResetContent();
CFtpFileFind ftpfind(pFtpConnection);
BOOL bfind=ftpfind.FindFile(NULL);
while(bfind)
{
 bfind=ftpfind.FindNextFile();
 CString strpath=ftpfind.GetFileURL();
 m_ftpfilelst.AddString(strpath);
}
}
```

(7) 按钮“上传”的实现函数，该函数实现将文件上传到 FTP 服务器中，代码如下：

```
void CFtpUploadDlg::OnUPLoad()
{
 CString str;
 CString strname;
 CFileDialog file(true, "file", NULL, OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
 "所有文件|*.*||", this); //构造文件对话框
 if(file.DoModal() == IDOK)
 {
 str=file.GetPathName(); //获得文件路径
 strname=file.GetFileName(); //获得文件名
 }
 if(bconnect)
 {
 BOOL bput=pFtpConnection->PutFile((LPCTSTR)str, (LPCTSTR)strname);
 if(bput){
 m_ftpfilelst.ResetContent();
 this->OnConnect();
 AfxMessageBox("上传成功");
 }
 }
}
```

(8) 单击复选框的实现函数，该函数用来设置是否使用匿名登录，代码如下：

```
void CFtpUploadDlg::OnNoname()
{
 int icheck=m_noname.GetCheck();
 if(icheck==1)
 {
 //设置控件不可用
 GetDlgItem(IDC_EDUSR)->EnableWindow(FALSE);
 GetDlgItem(IDC_EDPWD)->EnableWindow(FALSE);
 //设置显示文本
 GetDlgItem(IDC_EDUSR)->SetWindowText("anonymous");
 GetDlgItem(IDC_EDPWD)->SetWindowText("");
 }else
 {
 //设置控件可用
 GetDlgItem(IDC_EDUSR)->EnableWindow(TRUE);
 GetDlgItem(IDC_EDPWD)->EnableWindow(TRUE);
 //清空控件中数据
 GetDlgItem(IDC_EDUSR)->SetWindowText("");
 GetDlgItem(IDC_EDUSR)->SetWindowText("");
 }
}
```

## 举一反三

根据本实例，读者可以：

- 实现 FTP 文件的下载。

## 实例 371

HTTP 服务器多线程文件  
下载

本实例可以提高计算机安全性

实例位置：光盘\mingrisoft\13\371

## 实例说明

网上的许多下载软件都提供了多线程下载的功能，采用多线程，可以增加网络的吞吐量，提高文件的下载速度。本例实现了 HTTP 服务器多线程文件下载功能，实例运行结果如图 13.4 所示。

## 技术要点

实现多线程文件下载并不像想象中的那么复杂。具体思路如下。

首先确定下载的文件大小，根据文件大小确定采用几个线程下载，每个线程下载文件的不同部分；然后在线下载完后，将每个线程下载的数据按顺序组合为一个完整的文件。这样就实现了文件的多线程下载。

## 实现过程

(1) 新建一个基于对话框的应用程序。在对话框中添加静态文本控件、文本编辑控件、按钮控件、列表视图控件。

(2) 在对话框的头文件中引用“afxmt.h”、“afxinet.h”头文件，目的是使用 WinInet 类。

```
#include <afxmt.h>
#include <afxinet.h>
#pragma comment(lib, "wininet.lib")
```

(3) 在对话框类中定义如下成员变量：

```
UINT m_ThreadCount; //线程数量
DWORD m_PerFileSize; //每个线程下载的文件大小
DWORD m_EndFileSize; //最后一个线程下载的文件大小
HANDLE* m_pHthread; //线程句柄
void** m_pData; //每个线程下载数据到缓冲区,在每个线程均下载完成后合成一个完整的文件
CEvent* m_pEvent; //事件对象,用于线程同步
```

(4) 添加全局线程函数，实现下载任务。

```
//线程函数
DWORD WINAPI ThreadProc(LPVOID lpParameter)
{
 int index = *(int*)lpParameter;
 delete lpParameter;
 CInternetSession* pSession = new CInternetSession;
 CHttpConnection* pFtpCon = pSession->GetHttpConnection("127.0.0.1");
 CFtpDownloadDlg* pDlg = (CFtpDownloadDlg*)AfxGetMainWnd();
 CInternetFile* pFile = (CInternetFile*)pSession->OpenURL(pDlg->m_Dir);
 DWORD readsize;
 if (index < pDlg->m_ThreadCount-1)
 readsize = pDlg->m_PerFileSize;
 else
 readsize = pDlg->m_EndFileSize;
 pFile->Seek(index*pDlg->m_PerFileSize, FILE_BEGIN);
 pDlg->m_pData[index] = LocalAlloc(LMEM_FIXED, readsize);
 pFile->Read(pDlg->m_pData[index], readsize);
 delete pFile;
 delete pFtpCon;
 delete pSession;
 pDlg->m_pEvent[index].SetEvent();
 return 0;
}
```

(5) 处理“下载”按钮的单击事件，根据用户提供的网址下载文件。

```
void CFtpDownloadDlg::OnDownload()
{
 ...
}
```



图 13.4 HTTP 服务器多线程文件下载



```

UpdateData(TRUE);

if (m_Dir.IsEmpty())
{
 MessageBox("请输入下载路径");
 return;
}

CInternetSession * pSession = new CInternetSession;
CHttpConnection * pFtpCon = pSession->GetHttpConnection("127.0.0.1");
CInternetFile* pFile = (CInternetFile*)pSession->OpenURL(m_Dir,
1,INTERNET_FLAG_TRANSFER_BINARY|INTERNET_FLAG_RELOAD);

DWORD len = pFile->SeekToEnd();
//确定划分几个线程下载
//每个线程应下载的文件大小
m_PerFileSize= len / m_ThreadCount;
m_EndFileSize = m_PerFileSize+ len % m_ThreadCount;
delete pFile;
delete pFtpCon;
delete pSession;
int pos = m_Dir.ReverseFind('/');
//设置文件名
CString filename = m_Dir.Right(m_Dir.GetLength()-pos-1);
//插入字符串
m_List.InsertItem(m_List.GetItemCount(), "");
m_List.SetItemText(m_List.GetItemCount()-1, 0, filename);
//获得时间
CString time = CTime::GetCurrentTime().Format("%H:%M:%S");
m_List.SetItemText(m_List.GetItemCount()-1, 1, time);
m_List.SetItemText(m_List.GetItemCount()-1, 2, "C:\\\\"+filename);
for (int i = 0; i < m_ThreadCount; i++)
{
 int* temp = new int;
 *temp = i;
 m_pEvent[i].ResetEvent();
 m_pHthread[i] = CreateThread(NULL, 0, ThreadProc, temp, 0, NULL); //开启线程
}

for (int n = 0; n < m_ThreadCount; n++)
{
 WaitForSingleObject(m_pEvent[n], INFINITE);
}

CFile file ("c:\\\\"+filename, CFile::modeCreate|CFile::modeWrite); //创建文件
DWORD readsize;
for (int m = 0; m < m_ThreadCount; m++)
{
 if (m < m_ThreadCount-1)
 readsize = m_PerFileSize;
 else
 readsize = m_EndFileSize;
 file.Write(m_pData[m], readsize); //向文件写入数据
}
delete[] m_pData;
m_pData = new void*[m_ThreadCount];

```

### 举一反三

根据本实例，读者可以：

- 实现多 IP 文件下载。

## 实例 372 遍历 FTP 文件目录

本实例可以提高计算机安全性

实例位置：光盘\mingrisoft\13\372

### 实例说明

FTP 服务器多用来提供文件的下载。为了了解 FTP 服务器中的文件信息，有必要获取 FTP 文件目录。本例实现了遍历某个 FTP 服务器中的所有文件目录的功能，实例运行结果如图 13.5 所示。

## 技术要点

本例主要通过 CFtpFileFind 类获取 FTP 文件目录。首先创建一个 CInternetSession 对象,通过调用该对象的 GetFtp Connection 方法获得 CFtpConnection 类的一个指针,然后利用该指针构造 CFtpFileFind 类对象,最后调用 CFtpFileFind 类的 FindFile 和 FindNextFile 方法搜索目录。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加静态文本控件,设置其 Caption 属性为“FTP 服务器”、“端口”、“用户名称”和“用户密码”;添加一个按钮控件,设置其 Caption 属性为“登录”;添加一个树控件显示文件目录。

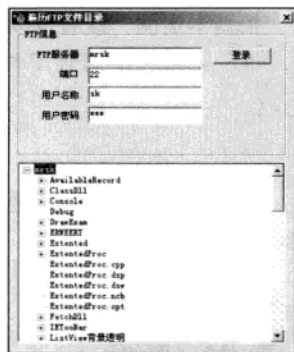


图 13.5 遍历 FTP 文件目录

- (3) 主要程序代码如下:

```
void CBrownFTPDirDlg::OnLogin()
{
 CString server;
 m_Server.GetWindowText(server); //获得服务器
 m_TreeInfo.DeleteAllItems(); //清空控件
 HTREEITEM hRoot = m_TreeInfo.InsertItem(server,0,0);
 ListDir("",hRoot);
}

void CBrownFTPDirDlg::ListDir(CString dir, HTREEITEM hParent)
{
 CString filename;
 CString server,port,user,pass;
 m_Server.GetWindowText(server);
 m_Port.GetWindowText(port);
 m_User.GetWindowText(user);
 m_Pass.GetWindowText(pass);
 CInternetSession session;
 CFtpConnection* pTemp = session.GetFtpConnection(server,user,pass,atoi(port));
 CFtpFileFind Find(pTemp);
 HTREEITEM hItem = hParent;
 HTREEITEM hSubItem;
 BOOL ret;
 if (dir.IsEmpty())
 ret = Find.FindFile(NULL,INTERNET_FLAG_EXISTING_CONNECT); //查找文件
 else
 ret = Find.FindFile(dir,INTERNET_FLAG_EXISTING_CONNECT);
 if (ret)
 {
 while (Find.FindNextFile()) //查找下一个文件
 {
 filename = Find.GetFileName();
 hSubItem = m_TreeInfo.InsertItem(filename,0,0, hParent); //插入节点

 if (Find.IsDirectory())
 {
 ListDir(dir+"\\"+filename,hSubItem);
 }
 }
 if (!Find.IsDirectory())
 {
 filename = Find.GetFileName();
 m_TreeInfo.InsertItem(filename,0,0,hItem);
 }
 else
 {
 ListDir(dir+"\\"+filename,hItem);
 }
 }
 Find.Close();
 delete pTemp;
}
```

## 举一反三

根据本实例,读者可以:

- 实现多 IP 文件下载。

## 13.3 邮件管理

电子邮件是网络数据通信中不可缺少的工具,本节通过几个实例介绍如何对电子邮件进行收取与发送。

## 实例 373 邮件接收程序

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\13\373

## 实例说明

电子邮件(E-mail)是通过电子通信系统进行信件的书写、发送和接收的一种方式。使用电子邮件可以加快信息的传递速度,并且可以传送文字、图像、声音等信息。本实例实现获取邮箱中邮件的主题和内容的功能,运行程序,输入“POP 服务器”、“用户名”、“密码”等信息后,单击“接收”按钮,邮箱中的邮件的主题和内容将显示在列表中,程序运行结果如图 13.6 所示。

## 技术要点

本实例通过 JMail 组件实现邮件的接收,JMail 组件可以实现邮件的接收和发送,邮件的接收需要通过 JMail 组件中的 IPOP3Ptr 指针和 IPOP3Ptr 指针对象的 Messages 指针完成,IPOP3Ptr 指针负责与邮件服务器的连接,建立连接后可以通过 IPOP3Ptr 指针对象的 Messages 指针来获得邮件的具体内容。IPOP3Ptr 指针的 Connect 方法实现与邮件服务器的连接,Connect 方法的参数包括 Username、Password、Server、Port,分别是登录的用户名、登录密码、POP 服务器、服务器端口。Messages 指针的 Count 属性能获得邮件的数量,Messages 指针的 Item 属性就是邮件文件。

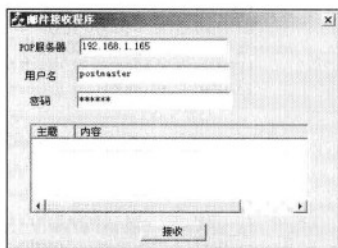


图 13.6 邮件接收程序

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加 3 个静态文本控件,设置 Caption 分别为“POP 服务器”、“用户名”和“密码”;添加 3 个文本编辑控件,设置 ID 属性分别为 IDC\_EDPOP、IDC\_EDUSER 和 IDC\_EDPWD;添加列表视图控件,设置 ID 属性为 IDC\_RECELST,View 属性设为 Report;添加成员变量 m\_recelst;添加 1 个按钮控件,设置 ID 属性为 IDC\_BTRECE,Caption 属性为“接收”。

(3) 添加“接收”按钮的实现函数 OnRece。

(4) 在 StdAfx.h 头文件中加入对 JMail 组件的引用,代码如下:

```
#import "jmail.dll"
using namespace jmail;
```

(5) 在 OnInitDialog 中初始化列表控件,代码如下:

```
BOOL CReceiveMailDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_recelst.SetExtendedStyle(LVS_EX_GRIDLINES); //设置扩展风格
 //设置列标题信息
```

```

m_recelst.InsertColumn(0,"主题",LVCFMT_LEFT,50);
m_recelst.InsertColumn(1,"内容",LVCFMT_LEFT,300);
m_recelst.InsertColumn(2,"时间",LVCFMT_LEFT,50);
HRESULT hr=::ColInitialize(NULL);
if(!SUCCEEDED(hr))
 return FALSE;
return TRUE;
}

```

(6) 按钮“接收”的实现函数，该函数实现邮件的接收，代码如下：

```

void CReceiveMailDlg::OnRece()
{
 m_recelst.DeleteAllItems();
 CString strpop;
 CString strusr;
 CString strpwd;
 //获得控件中数据
 GetDlgItem(IDC_EDPOP)->GetWindowText(strpop);
 GetDlgItem(IDC_EDUSER)->GetWindowText(strusr);
 GetDlgItem(IDC_EDPWD)->GetWindowText(strpwd);
 jmail::IMessagesPtr jmsg;
 jmail::IPOP3Ptr jpop;
 //设置POP3服务器
 jpop.CreateInstance(__uuidof(jmail::POP3));
 jmsg.CreateInstance(__uuidof(jmail::Message));
 try{
 jpop->Timeout=120;
 jpop->Connect((_bstr_t)strusr,(_bstr_t)strpwd,(_bstr_t)strpop,110);
 long mailcount=jpop->Messages->Count-1;
 for(long i=1;i<=mailcount;i++)
 {
 //设置邮件内容
 _bstr_t bssubject=jpop->Messages->Item[i]->Subject;
 _bstr_t bsbody=jpop->Messages->Item[i]->Body;
 COleDateTime time=jpop->Messages->Item[i]->Date;
 int count=m_recelst.InsertItem(i,"");
 m_recelst.SetItemText(count,0,(const char *)bssubject);
 m_recelst.SetItemText(count,1,(const char *)bsbody);
 m_recelst.SetItemText(count,2,(const char *)time.Format("%Y-%m-%d"));
 }
 }
 catch(_com_error e)
 {
 CString strerr;
 strerr.Format("错误信息: %s\r\n错误描述: %s",
 (LPCTSTR)e.ErrorMessage(), (LPCTSTR)e.Description());
 AfxMessageBox(strerr);
 return;
 }
 AfxMessageBox("接收完成");
}

```

(7) 在关闭窗口时关闭类库的引用，代码如下：

```

BOOL CReceiveMailDlg::DestroyWindow()
{
 ::CoUninitialize();
 return CDialog::DestroyWindow();
}

```

## 举一反三

根据本实例，读者可以：

- 实现定时收取电子邮件。

## 实例 374 发送电子邮件附件

本实例可以方便操作、提高效率

实例位置：光盘\mingrisoft\13\374

## 实例说明

本实例实现了发送带附件的电子邮件的功能。运行程序，在 SMTP 文本框中输入邮件服务



器的地址,在“收信人”文本框中输入收件人地址,单击“添加”按钮选择要发送的附件文件,最后通过“发送”按钮发送邮件,程序运行结果如图 13.7 所示。

### 技术要点

本实例主要通过 JMail 组件实现发送带附件的邮件,发送邮件需要使用 JMail 组件对象的 IMessagePtr 指针,该指针的 AddRecipient 方法用于添加收件人,Subject 方法用于设置邮件的主题,Body 方法用于设置邮件的内容,From 方法用于设置发件人,AddCustomAttachment 方法用于添加附件,Send 方法用于发送邮件。

### 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加 6 个静态文本控件,设置 Caption 属性分别为 SMTP、“发件人”、“收件人”、“主题”、“附件”和“内容”;添加 6 个文本编辑控件,设置 ID 属性分别为 IDC SMTP、IDC EDFROM、IDC EDRECE、IDC EDSUBJECT、IDC EDATT 和 IDC EDBODY;添加 2 个按钮,设置 ID 属性分别为 IDC\_BTADD 和 IDC\_BTSEND, Caption 属性分别为“添加”和“发送”。

(3) 添加“添加”按钮的实现函数 OnAdd,添加“发送”按钮的实现函数 OnBtsend。

(4) 在头文件 Stdafx.h 中添加对 JMail 组件的引用,代码如下:

```
#import "jmail.dll"
using namespace jmail;
```

(5) 在 OnInitDialog 中初始化类库,代码如下:

```
BOOL CSendMailWithAddDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 HRESULT hr::CoInitialize(NULL);
 if(!SUCCEEDED(hr))
 return FALSE;
 return TRUE; }

```

(6) 按钮“添加”的实现函数,该函数实现添加附件文件,代码如下:

```
void CSendMailWithAddDlg::OnAdd()
{
 CFileDialog file(true,"file",NULL,OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,
 "所有文件 (*.*)",this); //构造文件对话框
 if(file.DoModal()==IDOK)
 {
 CString str;
 str=file.GetPathName();
 GetDlgItem(IDC_EDATT)->SetWindowText(str);
 }
}

```

(7) 按钮“发送”的实现函数,该函数完成邮件的发送,代码如下:

```
void CSendMailWithAddDlg::OnBtsend()
{
 CString strserver;
 CString strrece;
 CString strsubject;
 CString strbody;
 CString stratt;
 CString strfrom;
 //获得控件中数据
 GetDlgItem(IDC_EDRECE)->GetWindowText(strrece);
 GetDlgItem(IDC_EDSUBJECT)->GetWindowText(strsubject);
 GetDlgItem(IDC_EDBODY)->GetWindowText(strbody);
 GetDlgItem(IDC_EDATT)->GetWindowText(stratt);
}

```

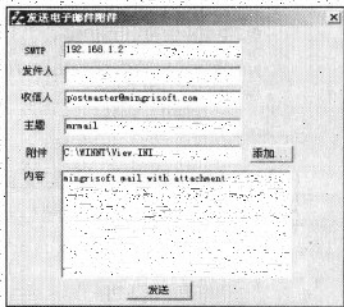


图 13.7 发送电子邮件附件

```

GetDlgItem(IDC_SMTP)->GetWindowText(strserver);
GetDlgItem(IDC_EDFROM)->GetWindowText(strfrom);
if(strfrom.IsEmpty())
{
 AfxMessageBox("请填写发信人地址");
 return;
}
if(strrece.IsEmpty())
{
 AfxMessageBox("请填写收信人地址");
 return;
}

jmail::IMessagePtr jmsg;
jmsg.CreateInstance(__uuidof(jmail::Message)); //设置接收人地址
jmsg->AddRecipient((LPCTSTR)strrece, "", "");
jmsg->Subject=(LPCTSTR)strsubject; //设置主题
jmsg->Body=(LPCTSTR)strbody; //设置邮件内容
jmsg->From=(LPCTSTR)strfrom; //设置发件人地址
jmsg->AddCustomAttachment((_bstr_t)stratt, (_bstr_t)"jmail", VARIANT_FALSE);
try{
 jmsg->Send((LPCTSTR)strserver, VARIANT_FALSE);
}
catch(_com_error e)
{
 CString strerr;
 strerr.Format("%s\r\n错误描述是%s", (LPCTSTR)e.ErrorMessage(),
 (LPCTSTR)e.Description());
 AfxMessageBox(strerr);
}
AfxMessageBox("发送成功");
}

```

(8) 在关闭窗口时关闭类库的引用, 代码如下:

```

BOOL CSendMailWithAddDlg::DestroyWindow()
{
 ::CoUninitialize();
 return CDialog::DestroyWindow();
}

```

## 举一反三

根据本实例, 读者可以:

- 实现批量发送电子邮件附件。

## 实例 375 使用 MAPI 发送邮件

本实例可以方便操作、提高效率

实例位置: 光盘\mingrisoft\13\375

## 实例说明

电子邮件具有传递速度快、使用方便等特点, 已成为人们交流信息、传输数据的主要方式。本例笔者利用 MAPI 函数实现了电子邮件的发送, 程序运行结果如图 13.8 所示。

## 技术要点

MAPI 函数位于 mapi32.dll 中, 在使用前, 需要将相关函数导入到程序中。本例共使用了 4 个 MAPI 函数, 分别为 MAPILogon、MAPISendMail、MAPIFreeBuffer 和 MAPILogoff。其中, MAPILogon 用于建立一个会话; MAPISendMail 用于发送电子邮件; MAPIFreeBuffer 用于释放邮件缓冲区里的数据; MAPILogoff 用于结束会话。

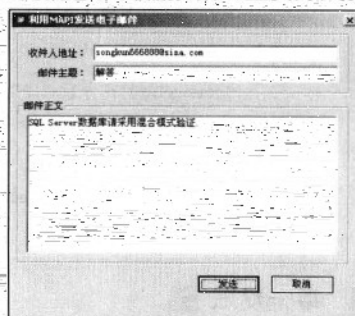


图 13.8 使用 MAPI 发送邮件

## 实现过程

(1) 新建基于对话框的应用程序。

(2) 在对话框上添加两个静态文本控件, 设置 Caption 属性为“收件人地址”和“邮件主题”, 添加一个文本编辑框控件; 添加两个按钮控件, 设置 Caption 属性为“发送”和“取消”。

(3) 主要程序代码。

```
void CSendEmailDlg::OnOK()
{
 result = LoadLibrary("mapi32.dll"); //加载mapi动态库
 //获取函数指针
 (FARPROC&)lpfnMAPILogon = GetProcAddress(result, "MAPILogon");
 (FARPROC&)lpfnMAPISendMail = GetProcAddress(result, "MAPISendMail");
 (FARPROC&)lpfnMAPIFreeBuffer = GetProcAddress(result, "MAPIFreeBuffer");
 (FARPROC&)lpfnMAPILogoff = GetProcAddress(result, "MAPILogoff");
 unsigned long a;
 lpfnMAPILogon(0, NULL, NULL, 0, 0, &a); //开始一个会话
 char* pcontext; //记录邮件正文
 CString str;
 m_content.GetWindowText(str);
 pcontext = str.GetBuffer(0);
 ULONG lresult;
 MapiMessage* m_messageInfo; //定义一个信息结构指针
 m_messageInfo = new MapiMessage;
 memset(m_messageInfo, 0, sizeof(MapiMessage)); //初始化m_messageInfo
 CTime time = CTime::GetCurrentTime(); //获取当前时间
 CString ctime = time.Format("%y/%m/%d/%H");
 char date[50];
 strcpy(date, ctime);
 CString subject;
 m_subject.GetWindowText(subject); //获取主题
 CString receiver, temp;
 m_ReceiverAddr.GetWindowText(temp); //获取收件人信息
 receiver = temp.Left(temp.Find('@')); //获取用户账户
 char addr1[100]; //记录用户账户
 char addr2[100]; //记录SMTP服务器
 strcpy(addr1, receiver);
 receiver = "SMTP:" + temp;
 strcpy(addr2, receiver);
 MapiRecipDesc m_recip = {0, MAPI_TO, addr1, addr2, 0, NULL}; //定义接收者信息
 m_messageInfo->lpszNoteText = pcontext; //设置邮件正文
 m_messageInfo->ulReserved = 0; //保留, 必须为0
 m_messageInfo->lpszSubject = subject.GetBuffer(0); //设置主题
 m_messageInfo->lpszDateReceived = date; //设置邮件发送时间
 m_messageInfo->lpszConversationID = NULL; //邮件所属线程一个字符串指针
 m_messageInfo->flFlags = MAPI_SENT; //邮件状态标记
 m_messageInfo->lpOriginator = NULL; //发送者信息
 m_messageInfo->nRecipCount = 1; //接收者人数
 m_messageInfo->nFileCount = 0; //附件数
 m_messageInfo->lpRecips = &m_recip; //设置接收者信息
 m_messageInfo->lpszMessageType = NULL; //邮件类型
 lresult = lpfnMAPISendMail(a, 0, m_messageInfo, MAPI_DIALOG, 0); //发送邮件
 if (lresult != SUCCESS_SUCCESS)
 {
 MessageBox("操作失败.", "提示", 64);
 }
 else
 {
 MessageBox("邮件发送成功.", "提示", 64);
 }
 m_content.Clear();
 delete m_messageInfo;
 lpfnMAPIFreeBuffer(m_messageInfo);
 lpfnMAPILogoff(a, 0, 0, 0);
 FreeLibrary(result);
}
```

## 举一反三

根据本实例, 读者可以:



实现批量发送电子邮件附件。

## 13.4 上网监控

对于企业网站管理人员,应该及时了解企业内部员工都浏览过哪些网站,如果有恶意网站,就应该根据具体情况对恶意网站进行过滤,本节通过实例介绍如何进行上网的监控。

## 实例 376 监控上网过程

这是一个可以提高基础技能的实例

实例位置:光盘\mingrisoft\13\376

## 实例说明

本实例实现了记录用户浏览过网站的网址的功能。运行程序,单击“开始监视”按钮后,程序将在系统托盘中运行,用户登录过的网站地址都记录在列表控件中。程序运行结果如图 13.9 所示。

## 技术要点

本实例的实现需要使用 SHDocVw 组件和 MSHTML 组件。SHDocVw 组件中的 IWebBrowser 2Ptr 指针主要对应着 IE 浏览器,MSHTML 组件中 IHTMLDocument2Ptr 指针主要对应着浏览器所浏览的内容。实例的实现过程主要是先通过 SHDocVw 组件的 IShellWindowsPtr 指针的 GetCount 方法获得浏览器的个数,再将 IShellWindowsPtr 指针的 Item 对象赋值给 IWebBrowser2Ptr 指针,然后通过 IWebBrowser2Ptr 指针的 GetDocument 方法获得 IHTMLDocument2Ptr 指针对象,最后通过 IHTMLDocument2Ptr 指针对象的 Geturl 方法获得浏览器地址栏的内容。

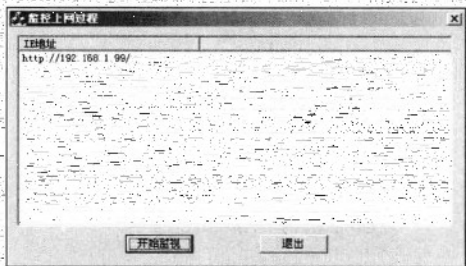


图 13.9 监控上网过程

## 实现过程

- (1) 新建基于对话框的应用程序。
- (2) 在对话框上添加列表视图控件,设置 ID 属性为 IDC\_ACTLIST,添加成员变量 m\_actlist,添加两个按钮控件,设置 ID 属性分别为 IDC\_BTENTER 和 IDC\_BTEXIT,设置 Caption 属性分别为“开始监视”和“退出”。
- (3) 在 NetProcessActDlg.h 文件中加入语句:

```
#include "atlbase.h"
#include <Mshhtml.h>
CCoPtr<IDispatch> spDispatch;
SHDocVw::IShellWindowsPtr m_spSHWinds;
```

- (4) 在 StdAfx.h 文件中加入语句:

```
#import "shdocvw.dll"
#import "mshhtml.tlb"
```

- (5) 在 OnInitDialog 中完成对列表控件的初始化,代码如下:

```
BOOL CNetProcessActDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_actlist.SetExtendedStyle(LVS_EX_GRIDLINES); //设置列表扩展风格
 m_actlist.InsertColumn(0, "IE地址", LVCFMT_LEFT, 200); //设置列标题
 ColInitialize(NULL); //初始化COM环境
 return TRUE;
}
```



(6) 设置定时器, 完成对浏览器中地址的监控, 代码如下:

```
void CNetProcessActDlg::OnTimer(UINT nIDEvent)
{
 KillTimer(1);
 BOOL bsame=FALSE;
 int n = m_spSHWinds->GetCount();
 for (int i = 0; i < n; i++){
 variant_t v = (long)i;
 IDispatchPtr spDisp = m_spSHWinds->Item(v);
 SHDocVw::IWebBrowser2Ptr spBrowser(spDisp);
 MSHTML::IHTMLDocument2Ptr pDoc2=spBrowser->GetDocument();//获得文档
 if(pDoc2!=NULL)
 {
 BSTR bsurl=pDoc2->Geturl();
 CString strurl=(CString)bsurl;
 int count=m_actlist.GetItemCount();
 for(int p=0;p<=count;p++)
 {
 CString itemstr=m_actlist.GetItemText(p,0);
 if(itemstr==strurl)
 {
 bsame=TRUE;
 goto end;
 }
 }
 if(bsame==FALSE)
 m_actlist.InsertItem(0,strurl,0);
 }
 }
 end:
 bsame=FALSE;
}

SetTimer(1,2000,NULL);
CDialog::OnTimer(nIDEvent);
}
```

(7) 按钮“开始监视”的实现函数, 该函数实现启动定时器的功能, 代码如下:

```
void CNetProcessActDlg::OnEnter()
{
 m_spSHWinds = NULL;
 m_spSHWinds.CreateInstance(__uuidof(SHDocVw::ShellWindows));
 SetTimer(1,2000,NULL);
}
```

### 举一反三

根据本实例, 读者可以:

- 实现对用户登录过的网页进行保存。

## 实例 377 网络监听工具

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\13\377

### 实例说明

在一个网段内, 如果网卡被设置为混合模式, 当网络中有数据报传输时, 无论数据报是否属于本机, 网卡都会接收到该数据报。本例利用该特点实现了一个网络监听工具, 实例运行结果如图 13.10 所示。

### 技术要点

要实现网络监听的功能, 用户需要对网络有一定的了解。在网络中, 数据是以帧的形式进行传输的。以 TCP 协议为例, 当用户发送数

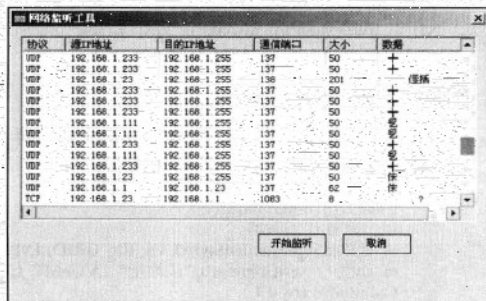


图 13.10 网络监听工具

据时,在传输层,用户数据的前端会附加TCP首部,TCP首部包括源端口号、目的端口号、序号、确认序号等信息,具体结构请参考本例实现过程中的HeadTCP结构。在网络层会附加IP首部,IP首部包括数据报的源地址和目标地址等信息,详细信息请参考本例实现过程中的HeadIP结构。在链路层附加地址解析协议和逆向地址解析协议,用于转换IP层和网络接口层使用的地址。

为了获得网络中传输的数据,首先需要创建一个原始套接字,该套接字获得的数据是IP层的数据报。包含IP首部、TCP或UDP首部、用户数据等信息。然后对获得的数据报去除IP首部,根据IP首部获得数据报的源地址、目的地址、采用的协议及数据报的长度等信息。接着根据不同的协议去除TCP或UDP首部,根据TCP或UDP首部确定源端口和目的端口。最后数据报剩余的部分即是用户数据。

## 实现过程

(1) 新建一个基于对话框的应用程序。在对话框中添加 Button、ListControl 控件。

(2) 在对话框的头文件中引用“winsock2.h”、“AFXSOCK.h”头文件。

```
#include "winsock2.h"
#pragma comment(lib, "ws2_32.lib")
#include "AFXSOCK.H"
```

(3) 在应用程序初始化时初始化套接字。

```
//初始化套接字
WSADATA data;
AfxSocketInit(&data);
```

(4) 在对话框头文件中定义IP首部、TCP首部、UDP首部等结构。

//定义IP数据报头结构, 20个字节

```
typedef struct HeadIP {
 unsigned char headerlen:4; //首部长度, 占4位
 unsigned char version:4; //版本, 占4位

 unsigned char servertype; //服务类型, 占8位, 即1个字节
 unsigned short totallen; //总长度, 占16位
 unsigned short id; //与idoff构成标识, 共占16位, 前3位是标识, 后13位

 //是片偏移
 unsigned short idoff;
 unsigned char ttl; //生存时间, 占8位
 unsigned char proto; //协议, 占8位
 unsigned short checksum; //首部校验和, 占16位
 unsigned int sourceIP; //源IP地址, 占32位
 unsigned int destIP; //目的IP地址, 占32位
}
```

}HEADIP;

//定义TCP数据报首部

```
typedef struct HeadTCP {
 WORD SourcePort; //16位源端口号
 WORD DePort; //16位目的端口
 DWORD SequenceNo; //32位序号
 DWORD ConfirmNo; //32位确认序号
 BYTE HeadLen; //与Flag为一个组成部分, 首部长度, 占4位, 保留6位, 6

 //位标识, 共16位
 BYTE Flag;
 WORD WndSize; //16位窗口大小
 WORD CheckSum; //16位校验和
 WORD UrgPtr; //16位紧急指针
}
```

}HEADTCP;

//定义UDP数据报首部

```
typedef struct HeadUDP {
 WORD SourcePort; //16位源端口号
 WORD DePort; //16位目的端口
 WORD Len; //16位UDP长度
 WORD ChkSum; //16位UDP校验和
}
```

}HEADUDP;

//定义ICMP数据报首部

```
typedef struct HeadICMP {
 BYTE Type; //8位类型
 BYTE Code; //8位代码
}
```



```
WORD ChkSum; //16位校验和
} HEADICMP;

//定义协议名称
struct PROTONAME{
 int value;
 char* protoname;
};
```

(5) 在对话框类的源文件中添加全局线程函数 ThreadFun, 根据原始套接字接收的数据, 逐一去除 IP 首部、TCP 首部、UDP 首部信息。

```
//线程函数
UINT ThreadFun(LPVOID pParam)
{
 CSniffAppDlg* pDlg = static_cast<CSniffAppDlg*>(pParam);

 MSG msg;
 char buffer[1000], sourceip[32], *tempbuf;
 char *ptemp;

 BYTE* pData = NULL; //实际数据报中的数据

 UINT sourceport;

 CString str;

 HEADIP* pHeadIP;
 HEADICMP* pHeadICMP;
 HEADUDP* pHeadUDP;
 HEADTCP* pHeadTCP;

 in_addr addr;

 int ret;
 while (TRUE)
 {
 pData = NULL;
 if (PeekMessage(&msg, pDlg->m_hWnd,
 WM_CLOSE, WM_CLOSE, PM_NOREMOVE))
 {
 closesocket(pDlg->m_Sock);
 break;
 }
 memset(buffer, 0, 1000);

 ret = recv(pDlg->m_Sock, buffer, 1000, 0);

 if (ret == SOCKET_ERROR)
 {
 continue;
 }
 else //接收到数据
 {
 tempbuf = buffer;

 pHeadIP = (HEADIP*)tempbuf;

 //获取数据报总长度
 WORD len = ntohs(pHeadIP->totallen);

 //获取源IP
 pDlg->m_List.InsertItem(pDlg->m_List.GetItemCount(), "");
 addr.S_un.S_addr = pHeadIP->sourceIP;
 ptemp = inet_ntoa(addr);

 pDlg->m_List.SetItemText(pDlg->m_List.GetItemCount()-1, 1, ptemp);

 //获取目的IP
 addr.S_un.S_addr = pHeadIP->destIP;
 ptemp = inet_ntoa(addr);
 pDlg->m_List.SetItemText(pDlg->m_List.GetItemCount()-1, 2, ptemp);

 //获取协议名称
 ptemp = get_protoname(pHeadIP->proto);
 strcpy(sourceip, ptemp);
```

```
pDlg->m_List.SetItemText(pDlg->m_List.GetItemCount()-1,0,sourceip);

//获取IP数据报总长度
WORD ipSumLen = ntohs(pHeadIP->totallen);

//IP数据报头总长度
int ipHeadLen = 20;

//获得去除IP层数据的长度
WORD netlen = ipSumLen - ipHeadLen;

//根据不同的协议获得不同协议的数据
switch (pHeadIP->proto)
{
case IPPROTO_ICMP:
 {
 pHeadICMP = (HEADICMP*)(tempbuf+20);

 pData = (BYTE*)(pHeadICMP)+4; //ICMP数据报头共4个字节
 //获取数据的长度
 netlen -= 4;

 break;
 }
case IPPROTO_UDP:
 {
 pHeadUDP = (HEADUDP*)(tempbuf+20);

 pData = (BYTE*)pHeadUDP+8; //UDP数据报头共8个字节

 sourceport = ntohs(pHeadUDP->SourcePort);

 str.Format("%d",sourceport);
 //设置源端口
 pDlg->m_List.SetItemText(pDlg->m_List.GetItemCount()-1,3,str);

 str.Empty();

 netlen -= 8;
 break;
 }
case IPPROTO_TCP:
 {
 pHeadTCP = (HEADTCP*)(tempbuf+20);
 sourceport = ntohs(pHeadTCP->SourcePort);

 pData = (BYTE*)pHeadTCP+20; //TCP数据报头共20个字节

 str.Format("%d",sourceport);
 //设置源端口
 pDlg->m_List.SetItemText(pDlg->m_List.GetItemCount()-1,3,str);
 str.Empty();
 netlen -= 20;
 break;
 }
}
//设置数据大小
str.Format("%d",netlen);
pDlg->m_List.SetItemText(pDlg->m_List.GetItemCount()-1,4,str);
str.Empty();
//设置数据
if (pData != NULL)
{
 str.Format(" %s",pData);
 pDlg->m_List.SetItemText(pDlg->m_List.GetItemCount()-1,5,str);
}
str.Empty();
}
return 0;
}
```

(6) 处理“开始监听”按钮的单击事件，监视网络中传输的数据。

```
void CSniffAppDlg::OnBeginlisten()
{
 //创建套接字
```



```
m_Sock = socket(AF_INET, SOCK_RAW, IPPROTO_IP);

char name[128];
memset(name, 0, 128);

hostent* phostent;

phostent = gethostbyname(name);

DWORD ip;

ip = inet_addr(inet_ntoa(*(in_addr*)phostent->h_addr_list[0]));

int timeout = 4000; //超时4秒

//设置接收数据的超时时间
setsockopt(m_Sock, SOL_SOCKET, SO_RCVTIMEO,
(const char*)&timeout, sizeof(timeout));

sockaddr_in skaddr;
skaddr.sin_family = AF_INET;
skaddr.sin_port = htons(700);
skaddr.sin_addr.S_un.S_addr = ip;
//绑定地址
if (bind(m_Sock, (sockaddr*)&skaddr, sizeof(skaddr)) == SOCKET_ERROR)
{
 MessageBox("地址绑定错误");
 return;
}

DWORD inBuffer=1;
DWORD outBuffer[10];
DWORD reValue = 0;

if (WSAIoctl(m_Sock, SIO_RCVALL, &inBuffer, sizeof(inBuffer),
&outBuffer, sizeof(outBuffer), &reValue, NULL, NULL) == SOCKET_ERROR)
{
 MessageBox("设置缓冲区错误.");
 closesocket(m_Sock);
 return;
}
else
 m_pThread = AfxBeginThread(ThreadFun, (void*)this);
}
```

### 举一反三

根据本实例，读者可以：

- 获取网络中传输的数据。

## 13.5 浏览器应用

浏览器是上网所需的必备工具，其不仅能浏览静态网页，而且还能解释动态网页。下面通过几个实例介绍浏览器的应用。

### 实例 378 制作自己的网络浏览软件

本实例可以美化界面、简化操作

实例位置：光盘\mingrisoft\13\378

### 实例说明

浏览器是上网所需的工具。本实例实现一个简单的网络浏览器。运行程序，在工具栏的文本框

中输入网站的地址，单击“浏览”按钮后网页就会在视图中显示，程序运行结果如图 13.11 所示。

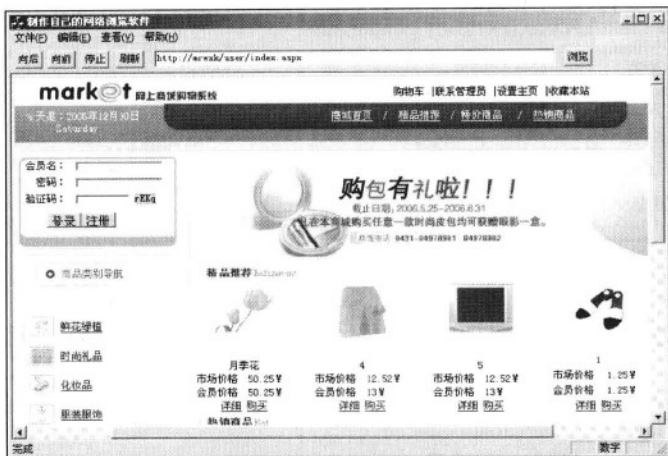


图 13.11 制作自己的网络浏览软件

## 技术要点

实例中使用了 CHtmlView 视图类，该视图类的使用和 WebBrowser 控件相似，基本上实现了 IE 浏览器的功能。CHtmlView 类的主要方法如表 13.1 所示。

表 13.1

CHtmlView 类的主要方法

| 方 法             | 描 述           |
|-----------------|---------------|
| GoBack          | 在浏览过的网页中向前浏览  |
| GoForward       | 在浏览过的网页中向后浏览  |
| GoHome          | 回到默认网页        |
| Navigate2       | 浏览指定网页        |
| Refresh         | 刷新当前网页        |
| Stop            | 停止数据的传输       |
| GetProperty     | 获得控件的属性       |
| GetTop          | 获得浏览器顶部在屏幕的位置 |
| SetOffline      | 设置为离线浏览       |
| GetLocationName | 获得浏览器标题的名称    |

实例中使用 CDialogBar 类的 Create 方法来创建工具条，该方法直接使用对话框资源来创建工具条，对话框窗体就是工具栏的窗体，用户只需设计对话框窗体即可，但此时对话框窗体的 Border 属性应为 None，Style 属性应为 Child。

## 实现过程

- (1) 新建一个基于单文档的应用程序，在创建过程中将 Cview 改为 CHtmlView。
- (2) 在工程中添加 Dialog 资源，设置 ID 属性为 IDD\_ADDRESS，Border 属性为 None，Style 属性为 Child，并在该资源中添加文本编辑控件，设置 ID 属性为 IDC\_EDADDRESS；添加 5 个按钮控件，设置 ID 属性分别为 IDC\_BTBACK、IDC\_BTFORWARD、IDC\_BTSTOP、IDC\_BTREFRESH 和 IDC\_RUN，Caption 属性分别为“向后”、“向前”、“停止”、“刷新”和“浏览”。
- (3) 在 MainFrm.h 文件中加入变量声明：



CDialogBar m\_DlgToolBar;

(4) 在 MainFrm.cpp 文件 OnCreate 函数中创建 CdialogBar 对象, 代码如下:

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
 ...//此处代码省略
 if (!m_DlgToolBar.Create(this, IDD_ADDRESS,
 CBRs_ALIGN_TOP, AFX_IDW_DIALOGBAR))
 {
 TRACE0("Failed to create dialogbar\n");
 return -1;
 }
 ...//此处代码省略
 return 0;
}
```

(5) 在头文件 MyBrowerView.h 中添加变量声明, 代码如下:

CString strsite;

(6) 浏览器工具栏上的各个按钮的实现函数, 完成向后、向前、停止、刷新、浏览的操作, 代码如下:

```
void CMyBrowerView::OnButtonsite()
{
 CClientDC dc(this);
 Navigate2(strsite, NULL, NULL);
 Invalidate(FALSE);
}

void CMyBrowerView::OnChangeEdaddress()
{
 CMainFrame *pw=(CMainFrame *)AfxGetMainWnd();
 CDialogBar *pb=&(pw->m_DlgToolBar);
 pb->GetDlgItemText(IDC_EDADDRESS, strsite);
}

void CMyBrowerView::OnButtonback()
{
 this->GoBack(); //后退
}

void CMyBrowerView::OnButtonfroward()
{
 this->GoForward(); //前进
}

void CMyBrowerView::OnButtonstop()
{
 this->Stop(); //停止
}

void CMyBrowerView::OnButtonrefresh()
{
 this->Refresh(); //刷新
}
```

## 举一反三

根据本实例, 读者可以:

- 设计可读取源程序的网页浏览器。

## 实例 379 XML 数据库文档的浏览

本实例可以提高数据库安全性

实例位置: 光盘\mingrisoft\13\379

## 实例说明

本实例主要实现对 XML 文件的读取。XML 是对 HTML 语言的一种扩展, XML 格式允许用户自定义标签, 通过自定义的标签很容易将数据组织起来, 形成记录集, 这样的记录集要比关系型数据库更具有灵活性。XML 现已被广泛的应用, 本实例实现了对 test.xml 文件的读取, 并将读取的结果显示在列表中。test.xml 文件的内容如下:

```
void CXMLViewDlg::OnRead()
{
```

```

unsigned short buff[128];
memset(buff,0,128);
MSXML::IXMLDOMDocumentPtr xdoc; //定义文档指针
xdoc.CreateInstance(__uuidof(MSXML::DOMDocument)); //实例化文档对象
xdoc->load("test.xml");
MSXML::IXMLDOMNodeListPtr nodelist=NULL;
nodelist=xdoc->selectNodes("database/filed");
MSXML::IXMLDOMNodePtr subnode;
long nodecount;
nodelist->get_length(&nodecount);
for(long i=0;i<nodecount;i++)
{
 subnode=nodelist->nextNode()->selectSingleNode((_bstr_t)"name");//查找指定节点
 _bstr_t bstrname=subnode->Gettext();
 m_xmllist.InsertItem(i,"");
 m_xmllist.SetItemText(i,0,bstrname);
}
nodelist->reset();
for(i=0;i<nodecount;i++)
{
 subnode=nodelist->nextNode()->selectSingleNode((_bstr_t)"type");
 _bstr_t bstrname=subnode->Gettext();
 m_xmllist.SetItemText(i,1,bstrname);
}
}
}

```

程序运行时读取 test.xml 文件中的内容，运行结果如图 13.12 所示。

## 技术要点

本实例主要通过 MSXML 组件实现。通过 MSXML 组件中 IXMLDOMDocumentPtr 指针调用 selectNodes 方法可以获得 MSXML 组件中 IXMLDOMNodeListPtr 指针对象，MSXML 组件中 IXMLDOMNodePtr 指针对象可以获得具体节点的值。selectNodes 方法主要是读取节点路径，XML 文件中  $\langle \rangle$  和  $\langle / \rangle$  称为一个节点，节点下面可以有子节点，例如 test.xml 文件中 filed 是 database 的子节点。节点中可以设置属性值，例如 test.xml 文件中 filed 节点中 id 就是属性。

IXMLDOMNodePtr 指针对象可以通过 IXMLDOMNodeListPtr 指针的 nextNode 对象的 selectSingleNode 方法获得，通过 IXMLDOMNodePtr 指针对象的 Gettext 方法获得节点内容。

## 实现过程

- (1) 新建名为 XMLView 的对话框 MFC 工程。
- (2) 在对话框上添加列表视图控件，设置 ID 属性为 XMLLIST，添加成员变量 m\_xmllist；添加两个按钮控件，设置 ID 属性分别为 IDC\_READ 和 IDC\_EXIT，Caption 属性分别为“读取”和“退出”。
- (3) 添加“读取”按钮的实现函数 OnRead；添加“退出”按钮的实现函数 OnExit。
- (4) 在文件 StdAfx.h 中加入对 msxml 组件的应用，代码如下：

```

#import "msxml.dll"
using namespace MSXML;

```

- (5) 在 OnInitDialog 中完成对列表控件的初始化，代码如下：

```

BOOL CXXMLViewDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 m_xmllist.SetExtendedStyle(LVS_EX_GRIDLINES);
 m_xmllist.InsertColumn(0,"name",LVCFMT_LEFT,70);
 m_xmllist.InsertColumn(1,"type",LVCFMT_LEFT,70);
 return TRUE;
}

```

- (6) 按钮“读取”的实现函数，该函数完成对 XML 文件的读取，代码如下：

```

void CXXMLViewDlg::OnRead()
{

```

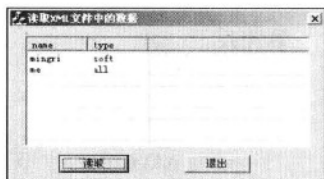


图 13.12 XML 数据库文档的浏览





```

unsigned short buff[128];
memset(buff,0,128);
HRESULT hr=::CoInitialize(NULL);
if(!SUCCEEDED(hr))
 return;
MSXML::IXMLDOMDocumentPtr xdoc; //定义文档指针
xdoc.CreateInstance(__uuidof(MSXML::DOMDocument)); //实例化文档
xdoc->load("test.xml"); //加载文档
MSXML::IXMLDOMNodeListPtr nodelist=NULL;
nodelist=xdoc->selectNodes(" database/filed");
MSXML::IXMLDOMNodePtr subnode;
long nodecount;
nodelist->get_length(&nodecount);
for(long i=0;i<nodecount;i++)
{
 subnode=nodelist->nextNode()->selectSingleNode((_bstr_t)"name");//查找指定节点
 _bstr_t bstname=subnode->Gettext(); //获得节点数据
 m_xmlist.InsertItem(i,"");
 m_xmlist.SetItemText(i,0,bstname);
 nodelist->reset();
 subnode=nodelist->nextNode()->selectSingleNode((_bstr_t)"type");
 bstname=subnode->Gettext();
 m_xmlist.SetItemText(i,1,bstname);
}
}

```

(7) 按钮“退出”的实现函数，该函数实现窗体的关闭，代码如下：

```

void CXMLViewDlg::OnExit()
{
 this->OnCancel();
}

```

### 举一反三

根据本实例，读者可以：

- 读写 XML 文件。
- 读写 XML 文件中节点属性值。

## 实例 380

### 使用 WebBrowser 执行脚本

本实例可以提高数据库安全性

实例位置：光盘\mingrisoft\13\380

### 实例说明

WebBrowser 可以通过接口进行许多操作，也可以执行已打开网页中的 Script 脚本。本实例在窗体上添加了一个浏览器控件，然后单击“确定”按钮即可执行网页中存在的 Script 脚本。程序运行如图 13.13 所示。

### 技术要点

本实例主要通过 WebBrowser 组件实现。通过该组件中 IHTMLDocument2Ptr 指针调用 GetScript 方法可以获得 Script 脚本函数并执行脚本代码，在使用 IHTMLDocument2Ptr 接口前应先载入 mshtml.tlb 库文件。载入代码如下：

```
#import "mshtml.tlb"
```

### 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框中添加 WebBrowser 的 ActiveX 控件，并创建名为 CWebBrowser2 的实体类，添加两个按钮控件，设置 Caption 属性为“确定”和“取消”。



图 13.13 使用 WebBrowser 执行脚本

(3) 在文件 StdAfx.h 中加入对 msxml 组件的应用, 代码如下:

```
void CExecScriptDlg::OnOK()
{
 MSHTML::IHTMLDocument2Ptr spDoc(m_webbrowser.GetDocument());
 if (spDoc)
 {
 IDispatchPtr spDisp(spDoc->GetScript()); // 获得脚本函数
 if (spDisp)
 {
 OLECHAR FAR* szMember = L"messagebox";
 DISPID dispid;
 HRESULT hr = spDisp->GetIDsOfNames(IID_NULL, &szMember, 1,
 LOCALE_SYSTEM_DEFAULT, &dispid);

 if (SUCCEEDED(hr))
 {
 COleVariant vtResult;
 static BYTE parms[] = VTS_BSTR;
 COleDispatchDriver dispDriver(spDisp, FALSE);
 dispDriver.InvokeHelper(dispid, DISPATCH_METHOD, VT_VARIANT,
 (void*)&vtResult, parms,
 "5+Math.sin(9)");
 }
 }
 }
}
```

(4) 所打开的 HTML 文件的源码如下:

```
<HTML>
<HEAD>
<TITLE>demo</TITLE>
<SCRIPT>
function messagebox(x)
{
 alert("hello word")
 return eval(x)
}
</SCRIPT>
</HEAD>
<BODY>
 使用WebBrowser执行脚本
</BODY>
</HTML>
```

## 举一反三

根据本实例, 读者可以:

- 用 WebBrowser 控件编辑 HTML 文件。

## 实例 381 电子书阅读器

本实例可以提高数据库安全性

实例位置: 光盘\mingrisoft\13\381

## 实例说明

现在可以在很多网站下载电子书进行阅读, 可以看到电子书能表现出现实中翻页的那种效果。当鼠标点击在电子书的右边时翻到下一页, 当点击到左边时翻到上一页, 在页面变化内容的时候可以看到翻页的立体效果, 运行结果如图 13.14 所示。

## 技术要点

在本示例中使用的主要技术又如何

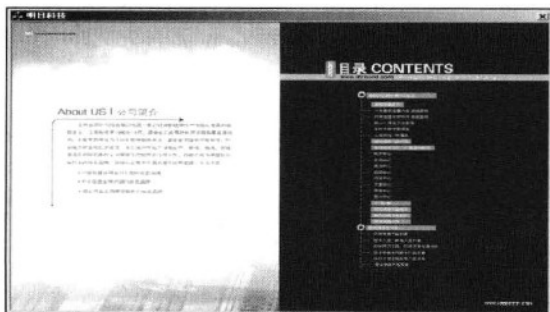


图 13.14 电子书阅读器

设定定时器、怎样将位图完全显示在窗口内。下面逐一进行介绍。

(1) 设置定时器。为了有翻页的效果,要在相同间隔的时间进行重复绘图,需要使用 SetTimer 函数来设置一个定时器。该语法格式如下:

```
UINT SetTimer(UINT nIDEvent, UINT nElapsed, void (CALLBACK EXPORT* lpfnTimer)(HWND, UINT, UINT, DWORD));
```

参数说明:

- nIDEvent: 指定了不为零的定时器标识符。
- nElapsed: 指定了定时值,以毫秒为单位。
- lpfnTimer: 指定了应用程序提供的 TimerProc 回调函数的地址,该函数被用于处理 WM\_TIMER 消息。如果这个参数为 NULL,则 WM\_TIMER 消息被放入应用程序的消息队列并由 CWnd 对象来处理。

本实例中设置定时器的代码如下:

```
SetTimer(1,150,NULL);
```

(2) 取消定时器的使用。当绘图操作完成时,需要调用 KillTimer 函数停止定时器的使用,销毁以前调用 SetTimer 创建的用 nIDEvent 标识的定时器事件。该语法格式如下:

```
BOOL KillTimer(int nIDEvent);
```

参数说明:

- nIDEvent: 传递给 SetTimer 的定时器事件值。

(3) 将位图完全显示在窗口内。要将一个位图完全的显示在窗口中要使用 StretchBlt 函数。该语法格式如下:

```
BOOL StretchBlt(int x, int y, int nWidth, int nHeight, CDC* pSrcDC, int xSrc, int ySrc, int nSrcWidth, int nSrcHeight, DWORD dwRop);
```

- x: 目标矩形左上角的 x 逻辑坐标。
- y: 目标矩形左上角的 y 逻辑坐标。
- nWidth: 目标矩形的宽度 (逻辑单位)。
- nHeight: 目标矩形的高度 (逻辑单位)。
- pSrcDC: 指定源设备上下文。
- xSrc: 源矩形左上角的 x 逻辑坐标。
- ySrc: 源矩形左上角的 y 逻辑坐标。
- nSrcWidth: 源矩形的宽度 (逻辑单位)。
- nSrcHeight: 源矩形的高度 (逻辑单位)。
- dwRop: 复制模式。

## 实现过程

(1) 创建一个单文档应用程序,工程名为“PictureOverturn”。

(2) 添加位图资源,添加完成后在 Resource.h 文件中查看位图的 ID 号,确保是连续的 ID 号。

(3) 在 CPictureOverturnView 类的头文件中声明变量,代码如下:

```
int m_BS_now; //用来保存当前位图的ID
int m_ID_next; //用来保存传递时下一张位图的ID
int m_ID_now; //用来保存传递时当前位图的ID
```

(4) 在 CPictureOverturnView 类的 PreCreateWindow 成员函数中添加的成员变量进行初始化操作,代码如下:

```
BOOL CPictureOverturnView::PreCreateWindow(CREATESTRUCT& cs)
{
 m_BS_now=IDB_BITMAP1;
 m_ID_now=IDB_BITMAP1;
 m_ID_next=IDB_BITMAP2;
 return CView::PreCreateWindow(cs);
}
```

(5) 在 CPictureOvertrun 类中,添加自定义函数 DrawPicture,用于根据位图的 ID 来绘制

位图，代码如下：

```
void CPictureOverturnView::DrawPicture(int ID_PICTURE)
{
 CDC *pDC= GetDC(); //得到客户区的设备环境
 CBitmap bitmap; //定义一个位图变量
 bitmap.LoadBitmap(ID_PICTURE); //将位图加载到位图变量中
 BITMAP bmp; //定义位图结构
 bitmap.GetBitmap(&bmp); //用位图信息填充位图结构
 CDC dcCompatible; //定义设备上下文对象
 dcCompatible.CreateCompatibleDC(pDC); //建立内存设备上下文与pDC设备上下文相匹配
 dcCompatible.SelectObject(&bitmap); //将位图选入内存设备上下文
 CRect rect;
 GetClientRect(rect); //得到客户区的大小

 pDC->StretchBlt(0,0,rect.Width(),rect.Height(),&dcCompatible,
 0,0,bmp.bmWidth,bmp.bmHeight,SRCCOPY); //将位图绘制在指定区域中
 ReleaseDC(pDC);
}
```

(6) 在 CPictureOverturnView 类中的 OnDraw 函数中添加代码对客户区进行绘制操作，调用 DrawPicture 绘制第一幅位图，代码如下：

```
void CPictureOverturnView::OnDraw(CDC* pDC)
{
 CPictureOverturnDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 DrawPicture(m_BS_now); //根据m_BS_now存储的位图ID进行绘制位图
}
```

(7) 处理 CPictureOvertureView 的 WM\_LBUTTONDOWN 消息，在该事件的处理函数 OnLButtonDown 中，进行判断用户鼠标点击的位置，如果是在窗口客户区的左边是翻向前一页，点击在客户区的右边则是翻向下一页，代码如下：

```
void CPictureOverturnView::OnLButtonDown(UINT nFlags, CPoint point)
{
 static int i=IDB_BITMAP1; //定义一个常量用来标记位图的ID
 CRect ClientRect;
 GetClientRect(ClientRect); //得到客户区的大小
 int Clientwidth=ClientRect.Width(); //得到客户区的宽度
 Clientwidth /=2;
 CString str;
 //判断鼠标的位置
 if(point.x>Clientwidth) //鼠标点击在客户区的右边
 {
 if(i==IDB_BITMAP1+16)
 {
 i=IDB_BITMAP1+16; //位图达到最后一页
 }
 else
 {
 int picture_pos=i;
 //传递图片的ID，并且在内部调用SetTimer函数
 DrawLToR(picture_pos,picture_pos+1);
 i++; //页面加一
 m_BS_now++; //保存当前的页面，放大缩小时使用
 }
 }
 else //鼠标点击在客户区的左边
 {
 if(i==IDB_BITMAP1)
 {
 i=IDB_BITMAP1;
 }
 else
 {
 int picture_pos=i;
 //自定义函数，根据ID调用SetTimer函数
 DrawRToL(picture_pos,picture_pos-1);
 i--;
 m_BS_now--;
 }
 }
 CView::OnLButtonDown(nFlags, point);
}
```

(8) 在 CPictureOvertureView 类中，添加自定义函数 DrawLToR，用于传递位图 ID，设置



的 1 号定时器调用 SetTimer 函数, 代码如下:

```
void CPictureOvertureView::DrawLToR(int IDNOW, int IDNEXT)
{
 m_ID_now=IDNOW; //将ID赋值给m_ID_now变量
 m_ID_next= IDNEXT;
 SetTimer(1,150,NULL); //设定1号定时器
}
```

(9) 在 CPictureOvertureView 类中, 添加自定义函数 DrawRToL, 用于传递位图 ID, 设置的 3 号定时器调用 SetTimer 函数, 代码如下:

```
void CPictureOvertureView::DrawRToL(int IDNOW, int IDNEXT)
{
 m_ID_now=IDNOW;
 m_ID_next= IDNEXT;
 SetTimer(3,150,NULL); //设定3号定时器
}
```

(10) 处理 CPictureOvertureView 类的 WM\_TIMER 消息, 在该事件的处理函数 OnTimer 中, 参数 nIDEvent 代表着定时器的序号, 所以可以根据定时器不同的序号使其进行不同的操作, 代码如下:

```
switch(nIDEvent)
{
case 1: //图片由左向右翻, 左页的翻转(第一步)
{
 OverPicture_LToR();
 break;
}
case 2: //图片由左向右翻, 右页的翻转 (第二步)
{
 OverPicture_LToR2();
 break;
}
case 3:
{
 OverPicture_RToL();
 break;
}
case 4:
{
 OverPicture_RToL2();
 break;
}
default:
{
 break;
}
}
CView::OnTimer(nIDEvent);
}
```

(11) 在 CPictureOvertureView 类中, 添加自定义函数 OverPicture\_LToR, 用于进行页面向下一页翻页的效果, 在其中定义一个静态变量用来记录翻页的程度, 此函数被 1 号定时器多次的调用, 并在最后调用 SetTimer 函数设定 2 号定时器, 代码如下:

```
void CPictureOvertureView::OverPicture_LToR()
{
 static int page_step=0; //用来记录翻页的程度
 CDC *pDC= GetDC();
 CBitmap bitmapnow,bitmapnext; //bitmapnow是当前的位图, bitmap是下一张位图
 bitmapnow.LoadBitmap(m_ID_now);
 bitmapnext.LoadBitmap(m_ID_next);
 BITMAP bmpnow, bmpnext; //定义bitmapnow位图的结构
 bitmapnow.GetBitmap(&bmpnow); //当前的位图结构
 bitmapnext.GetBitmap(&bmpnext); //下一张的位图结构
 CDC dcnow;
 dcnow.CreateCompatibleDC(pDC); //将当前图片选入设备描述表中
 CDC dcnext;
 dcnext.CreateCompatibleDC(pDC); //将下一张图片选入设备描述表中
 dcnext.SelectObject(&bitmapnext);
 CRect rect;
 GetClientRect(rect);

 //先画出左边不动的当前位图
 pDC->StretchBlt(0,0,rect.Width()/2,rect.Height(),&dcnow,
```

```

0,0,bmpnow.bmWidth/2,bmpnow.bmHeight,SRCCOPY);
//画出右边下一张的位图
pDC->StretchBlt(rect.Width()-rect.Width()/12*page_step, //填入矩形的x点
0, //填入矩形的y点
rect.Width()/12*page_step, //填入矩形的宽度
rect.Height(), //填入矩形的高度
&dcnext, //存放位图的设备描述表
bmpnext.bmWidth/2+bmpnext.bmWidth/2-bmpnext.bmWidth/12*page_step, //位图的起始x点
0, //位图的起始y点
bmpnext.bmWidth/12*page_step, //位图的宽度
bmpnext.bmHeight, //位图的高度
SRCCOPY); //复制的方式
//画出右边当前变形的位图, 覆盖了下一张位图
pDC->StretchBlt(rect.Width()/2,0,rect.Width()/2-rect.Width()
/12*page_step,rect.Height(),&dcnow,
bmpnow.bmWidth/2,0,bmpnow.bmWidth/2,bmpnow.bmHeight,SRCCOPY);
page_step++; //增加步数
if(page_step>6)
{
 page_step=0;
 KillTimer(1); //取消1号定时器的调用
 ReleaseDC(&dcnow); //释放内存设备上下文
 ReleaseDC(&dcnext);
 ReleaseDC(pDC);
 SetTimer(2,150,NULL); //调用2号定时器
}
ReleaseDC(&dcnow);
ReleaseDC(&dcnext);
ReleaseDC(pDC);
}

```

(12) 在 CPictureOvertureView 类中, 添加自定义函数 OverPicture\_LToR2, 用于进行页面向下一页翻页效果的第二步, 此函数被 2 号定时器中多次调用, 当调用次数达到设定数目, 则取消 2 号定时器的调用, 代码如下:

```

void CPictureOvertureView::OverPicture_LToR2()
{
 static int page_step=0;
 CDC *pDC = GetDC();
 CBitmap bitmapnow,bitmapnext; //bitmapnow是当前的位图, bitmap是下一张位图
 bitmapnow.LoadBitmap(m_ID_now);
 bitmapnext.LoadBitmap(m_ID_next);
 BITMAP bmpnow ,bmpnext;
 bitmapnow.GetBitmap(&bmpnow);
 bitmapnext.GetBitmap(&bmpnext);
 CDC dcnow;
 dcnow.CreateCompatibleDC(pDC);
 dcnow.SelectObject(&bitmapnow);
 CDC dcnext;
 dcnext.CreateCompatibleDC(pDC);
 dcnext.SelectObject(&bitmapnext);
 CRect rect;
 GetClientRect(rect);
 //先画出右边不动的下一张位图
 pDC->StretchBlt(rect.Width()/2,0,rect.Width()/2,rect.Height(),&dcnext,
 bmpnext.bmWidth/2,0,bmpnext.bmWidth/2,bmpnext.bmHeight,SRCCOPY);
 //画出左边当前的位图
 pDC->StretchBlt(
 0, //填入矩形的x点
 0, //填入矩形的y点
 rect.Width()/2-rect.Width()/12*page_step, //填入矩形的宽度
 rect.Height(), //填入矩形的高度
 &dcnow, //存放位图的设备描述表
 0, //位图的起始x点
 0, //位图的起始y点
 bmpnow.bmWidth/2-bmpnow.bmWidth/12*page_step, //位图的宽度
 bmpnow.bmHeight, //位图的高度
 SRCCOPY); //复制的方式
 //画出左边下一张变形的位图, 覆盖了当前位图
 pDC->StretchBlt(rect.Width()/2-rect.Width()/12*page_step, //矩形的x点
 0, //矩形的y点
 rect.Width()/12*page_step, //矩形的宽度
 rect.Height(),
 &dcnext,
 0, //位图的x点
 0, //位图的y点
 bmpnext.bmWidth/2,
 bmpnext.bmHeight,
 SRCCOPY);
}

```

```

page_step++;
page_step++;
if(page_step>6)
{
 page_step=0;
 KillTimer(2);
 ReleaseDC(&dcnow);
 ReleaseDC(&dcnext);
 ReleaseDC(pDC);
 DrawPicture(m_BS_now); //再画一遍当前的图片, 这样效果会好一些
}
ReleaseDC(&dcnow);
ReleaseDC(&dcnext);
ReleaseDC(pDC);

```

(13) 在 CPictureOverturnView 类中, 添加自定义函数 OverPicture\_RTOL, 用于实现向前一页翻页效果的第一步, 这一部分效果完成时会设定 4 号定时器, 进行调用翻页效果的第二部分, 代码如下:

```

void CPictureOverturnView::OverPicture_RTOL()
{
 static int page_step=0;
 CDC *pDC= GetDC();
 CBitmap bitmapnow,bitmapnext; //bitmapnow是当前的位图, bitmapnext是上一张位图
 bitmapnow.LoadBitmap(m_ID_now); //当前页
 bitmapnext.LoadBitmap(m_ID_next); //前一页
 BITMAP bmpnow ,bmpnext; //定义bitmapnow位图的结构
 bitmapnow.GetBitmap(&bmpnow); //当前的位图结构
 bitmapnext.GetBitmap(&bmpnext); //下一张的位图结构
 CDC dcnow;
 dcnow.CreateCompatibleDC(pDC);
 dcnow.SelectObject(&bitmapnow); //将当前图片选入设备描述表中
 CDC dcnext;
 dcnext.CreateCompatibleDC(pDC);
 dcnext.SelectObject(&bitmapnext); //将下一张图片选入设备描述表中
 CRect rect;
 GetClientRect(rect);
 //先画出右边不动的当前位图
 pDC->StretchBlt(rect.Width()/2,0,rect.Width()/2,rect.Height(),&dcnow,
 bmpnow.bmWidth/2,0,bmpnow.bmWidth/2,bmpnow.bmHeight,SRCCOPY);
 //画出右边下一张的位图
 pDC->StretchBlt(0, //填入矩形的x点
 0, //填入矩形的y点
 rect.Width()/12*page_step, //填入矩形的宽度
 rect.Height(), //填入矩形的高度
 &dcnext, //存放位图的设备描述表
 0, //位图的起始x点
 0, //位图的起始y点
 bmpnext.bmWidth/12*page_step, //位图的宽度
 bmpnext.bmHeight, //位图的高度
 SRCCOPY); //复制的方式

 //画出右边当前变形的位图, 覆盖了下一张位图
 pDC->StretchBlt(rect.Width()/12*page_step,0,rect.Width()
 /2-rect.Width()/12*page_step,rect.Height(),&dcnow,
 0,0,bmpnow.bmWidth/2,bmpnow.bmHeight,SRCCOPY);
 page_step++;
 if(page_step>6)
 {
 page_step=0;
 KillTimer(3);
 ReleaseDC(&dcnow);
 ReleaseDC(&dcnext);
 ReleaseDC(pDC);
 SetTimer(4,150,NULL); //调用4号定时器
 }
 ReleaseDC(&dcnow);
 ReleaseDC(&dcnext);
 ReleaseDC(pDC);
}

```

(14) 在 CPictureOverturnView 类中, 添加自定义函数 OverPicture\_RTOL2, 用于向前一页翻页效果的第二步, 完成向前一页翻页效果, 最后取消 4 号定时器调用, 并且绘制当前位图使显示效果更好, 代码如下:

```

void CPictureOverturnView::OverPicture_RTOL2()
{
 static int page_step=0;
 CDC *pDC= GetDC();

```

```

CBitmap bitmapnow,bitmapnext;//bitmapnow是当前的位图, bitmapnext是上一张位图
bitmapnow.LoadBitmap(m_ID_now); //大的
bitmapnext.LoadBitmap(m_ID_next); //小的
BITMAP bmpnow, bmpnext; //定义bitmapnow位图的结构
bitmapnow.GetBitmap(&bmpnow); //当前的位图结构
bitmapnow.GetBitmap(&bmpnext); //下一张的位图结构
CDC dcnow;
dcnow.CreateCompatibleDC(pDC);
dcnow.SelectObject(&bitmapnow); //将当前图片选入设备描述表中
CDC dcnext;
dcnext.CreateCompatibleDC(pDC);
dcnext.SelectObject(&bitmapnext); //将下一张图片选入设备描述表中
CRect rect;
GetClientRect(rect);
//先画出左边不动的当前位图
pDC->StretchBlt(0,0,rect.Width()/2,rect.Height(),&dcnext,
0,0,bmpnext.bmWidth/2,bmpnext.bmHeight,SRCCOPY);
//画出右边下一张的位图
pDC->StretchBlt(
 rect.Width()/2+rect.Width()/12*page_step, //填入矩形的x点
 0, //填入矩形的y点
 rect.Width()/2-rect.Width()/12*page_step, //填入矩形的宽度
 rect.Height(), //填入矩形的高度
 &dcnow, //存放位图的设备描述表
 bmpnow.bmWidth/2+bmpnow.bmWidth/12*page_step, //位图的起始x点
 0, //位图的起始y点
 bmpnow.bmWidth/2-bmpnow.bmWidth/12*page_step, //位图的宽度
 bmpnow.bmHeight, //位图的高度
 SRCCOPY); //复制的方式
//画出右边当前变形的位图, 覆盖了下一张位图
pDC->StretchBlt(rect.Width()/2,
0,
rect.Width()/12*page_step,
rect.Height(),
&dcnext,
bmpnext.bmWidth/2,
0,
bmpnext.bmWidth/2,
bmpnext.bmHeight,
SRCCOPY);
//填入矩形的x点
//填入矩形的y点
//填入矩形的宽度
//填入矩形的高度
//存放位图的设备描述表
//位图的起始x点
//位图的起始y点
//位图的宽度
//位图的高度
//复制的方式
page_step++;
if(page_step>6)
{
 page_step=0;
 KillTimer(4);
 ReleaseDC(&dcnow);
 ReleaseDC(&dcnext);
 ReleaseDC(pDC);
 DrawPicture(m_BS_now); //再画一遍当前的图片, 这样效果会好一些
}
ReleaseDC(&dcnow);
ReleaseDC(&dcnext);
ReleaseDC(pDC);
}

```

## 举一反三

根据本实例, 读者可以:

- 电子相册屏幕保护程序;
- 图片百叶窗效果。

## 13.6 网上信息提取

网上资源是一个大宝库, 合理利用网上资源, 可以提升企业竞争力。下面通过几个实例介绍如何提取和利用网上信息。



## 实例 382 定时提取网页源码

这是一个可以启发思维的实例

实例位置: 光盘\mingrisoft\13\382

## 实例说明

通过获取网页的源代码, 可以对网页进行分析及修改, 以提取有价值的信息。本实例实现了每隔一定时间读取网页源代码的功能。运行程序, 在“设置时间”文本框输入读取源码的时间间隔, 在“网页地址”文本框输入将要查看源码的网页地址, 单击“设置”按钮, 程序会在系统托盘中运行, 并将获得的网页源码显示在程序的编辑框中。程序运行结果如图 13.15 所示。

## 技术要点

本实例主要通过 CInternetSession 类和 CHttpFile 类实现。通过 CInternetSession 类的构造函数建立一个连接会话, 然后通过 CInternetSession 类的 OpenURL 方法获得 CHttpFile 对象, 最后通过 ReadString 方法将源码读取出来。

## 实现过程



图 13.15 定时提取网页源码

(1) 新建名为 GetWebSource 的对话框 MFC 工程。

(2) 在对话框上添加按钮控件, 设置 ID 属性为 IDC\_BTGET, Caption 属性为“设置”; 添加 3 个文本编辑控件, 设置 ID 属性分别为 IDC\_EDTIMER、IDC\_EDADDRESS 和 IDC\_EDWEBSOURCE, 设置 ID 属性为 IDC\_EDWEBSOURCE 的文本编辑控件的 Horizontal Scroll 和 Vertical Scroll 属性。

(3) 在头文件 GetWebSourceDlg.h 中添加变量声明:

```
CString strtime;
```

(4) 按钮“设置”的实现函数, 该函数完成定时器的启动, 代码如下:

```
void CGetWebSourceDlg::OnGet()
{
 GetDlgItem(IDC_EDTIMER)->GetWindowText(strtime);
 this->SetTimer(1, atoi(strtime), NULL);
}
```

(5) 设置定时器, 完成对网页源代码的获取, 代码如下:

```
void CGetWebSourceDlg::OnTimer(UINT nIDEvent)
{
 KillTimer(1);
 CString straddress;
 GetDlgItem(IDC_EDADDRESS)->GetWindowText(straddress); //获得网址
 CInternetSession mySession(NULL, 0);
 CHttpFile* myHttpFile=NULL;
 CString strsource, strline;
 myHttpFile=(CHttpFile*)mySession.OpenURL(straddress); //打开网址
 while(myHttpFile->ReadString(strline)) //读取字符串
 {
 strsource+=strline;
 strsource+="\r\n";
 }
 myHttpFile->Close;
 mySession.Close;
 GetDlgItem(IDC_EDWEBSOURCE)->SetWindowText(strsource); //显示源码
 SetTimer(1, atoi(strtime), NULL);
 CDialog::OnTimer(nIDEvent);
}
```

(6) 在关闭窗口时关闭定时器, 代码如下:

```
BOOL CGetWebSourceDlg::DestroyWindow()
{
 KillTimer(1);
 return CDialog::DestroyWindow();
}
```

### 举一反三

根据本实例, 读者可以:

- 获取网页中的指定内容。

## 实例 383 网上天气预报

这是一个可以提高基础技能的实例

实例位置: 光盘\mingrisoft\13\383

### 实例说明

现在许多网站都提供了天气预报服务, 用户可以随时得到天气信息。本实例实现了从网页中提取天气预报信息的功能。运行程序, 选择列表中的城市后, 单击“获取”按钮后, 该城市的天气预报信息将显示在列表中。程序运行结果如图 13.16 所示。

### 技术要点

本实例主要通过 CHttpFile 类的 ReadString 方法来获取网页的源代码, ReadString 方法是逐行的获取源代码, 实例中每读取一行代码都要和固定的字符进行比较, 固定字符可以设为城市的名称, 也可以设为“天气”、“气温”等。找到固定字符后根据天气预报信息在网页源代码中所在行与固定字符所在行的间隔行数来提取天气预报信息。

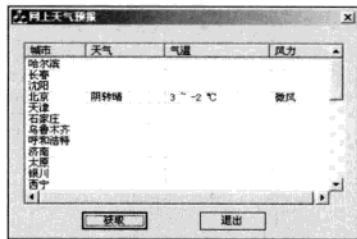


图 13.16 网上天气预报

### 实现过程

- (1) 新建名为 GetAirReport 的对话框 MFC 工程。
- (2) 在对话框上添加一个列表视图控件, 设置 ID 属性为 IDC\_REPORTLIST, 添加成员变量 m\_reportlist; 添加两个按钮控件, 设置 ID 属性分别为 IDC\_BTGET 和 IDC\_BTEXIT。
- (3) 主要程序代码。

```
void CGetAirReportDlg::OnGet()
{
 BOOL bNext1=FALSE, bNext2=FALSE, bNext3=FALSE;
 int leftpos=0; //取某行源码 "</TD>" 的位置
 int rightpos=0; //取某行源码 ">" 的位置
 int isel=m_reportlist.GetSelectionMark();
 if(isel<0)
 {
 AfxMessageBox("请选择城市");
 return;
 }
 address.Format("%s/%s.html", addressfront, city2[isel][1]);
 strtmp1.Format("city\">%s", city2[isel][0]);
 CString strsource;
 CInternetSession mySession(NULL, 0);
 CHttpFile* myHttpFile=NULL;
 myHttpFile=(CHttpFile*)mySession.OpenURL(address);
 while(myHttpFile->ReadString(strsource))
 {
 //如果某行源码中有提取的字符, 开始进行处理
 if(strsource.Find(strtmp1)>0)
 {
 bNext1=TRUE;
 if(bNext1)
 {

```

```

int leftpos=strsource.Find("map-layer-weaheer");

if(leftpos>0)
{
 strtmp2=strsource.Right(strsource.GetLength()-strlen("map-layer-weaheer")-leftpos-2);
 rightpos=strtmp2.Find("</div>");
 strtmp2=strtmp2.Left(rightpos);
 strweather=strtmp2;
 bNext1=FALSE;
 bNext2=TRUE;
}
}
if(bNext2)
{
 int leftpos=strsource.Find("map-layer-temp");
 if(leftpos>0)
 {
 strtmp2=strsource.Right(strsource.GetLength()-strlen("map-layer-temp")-leftpos-2);
 rightpos=strtmp2.Find("</div>");
 strtmp2=strtmp2.Left(rightpos);
 strtemperature=strtmp2;
 bNext2=FALSE;
 bNext3=TRUE;
 }
}
if(bNext3)
{
 int leftpos=strsource.Find("map-layer-wind");
 if(leftpos>0)
 {
 strtmp2=strsource.Right(strsource.GetLength()-strlen("map-layer-wind")-leftpos-2);
 rightpos=strtmp2.Find("</div>");
 strtmp2=strtmp2.Left(rightpos);
 strwind=strtmp2;
 bNext3=FALSE;
 goto end;
 }
}

end:
myHttpFile->Close();
mySession.Close();
m_reportlist.SetItemText(isel,1,strweather);
m_reportlist.SetItemText(isel,2,strtemperature);
m_reportlist.SetItemText(isel,3,strwind);
}

```

### 举一反三

根据本实例，读者可以：

- 获取网页上固定信息。

## 实例 384 网页链接提取器

这是一个可以提高基础技能的实例

实例位置：光盘\mingrisoft\13\384

### 实例说明

网上的许多下载软件都提供了提取网页链接的功能，当用户打开一个网页时，能够获得该网页的所有链接，用以搜索下载的文件资源。本实例现了提取网页链接的功能。实例运行结果如图 13.17 所示。

### 技术要点

Windows 系统中的 shdocvw.dll 动态库提供了与



图 13.17 网页链接提取器

网页操作有关的类。为了使用这些类，需要导入 shdocvw.dll 动态库。在 StdAfx.h 头文件中添加如下语句导入 shdocvw.dll 动态库。

```
#import <shdocvw.dll>
```

编译应用程序，系统会生成 shdocvw.tlh 文件，在该文件中定义了一个命名空间 SHDocVw，其中包含了操作网页的接口和类。

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在对话框中添加列表视图控件、按钮控件。
- (3) 在 StdAfx.h 头文件中导入 shdocvw.dll 动态库。

```
#import <shdocvw.dll>
```

- (4) 在对话框类的源文件中引用 atlbase.h、Mshtml.h 和 comdef.h 头文件。

```
#include <atlbase.h>
#include <Mshtml.h>
#include "comdef.h"
```

- (5) 在对话框头文件中定义一个 SHDocVw::IshellWindowsPtr 接口变量 m\_pSHWnd。

```
SHDocVw::IshellWindowsPtr m_pSHWnd;
```

- (6) 处理“查看”按钮的单击事件，查看浏览器当前页面的网页链接。

```
void CFetchPageDlg::OnLookup()
{
 TCHAR host[MAX_COMPUTER];
 CComPtr<IDispatch> spDispatch;
 CComQIPtr<IHTMLDocument2, &IID_IHTMLDocument2> pDoc2;
 CComPtr<IHTMLElementCollection> pElementCol;
 CComPtr<IHTMLAnchorElement> pLoct;
 int rowcount = m_List.GetItemCount();
 for (int i = 0; i < rowcount; i++)
 {
 IWebBrowser2 *pBrowser = (IWebBrowser2 *)m_List.GetItemData(i);
 if (pBrowser)
 {
 pBrowser->Release();
 }
 }
 m_List.DeleteAllItems();
 m_Num = 0;
 if (m_pSHWnd)
 {
 int n = m_pSHWnd->GetCount();
 for (int i = 0; i < n; i++)
 {
 variant t v = (long)i;
 IDispatchPtr spDisp = m_pSHWnd->Item(v);
 SHDocVw::IWebBrowser2Ptr spBrowser(spDisp);
 if (spBrowser)
 {
 if (SUCCEEDED(spBrowser->get_Document(&spDispatch)))
 {
 pDoc2 = spDispatch;
 if (pDoc2 != NULL)
 {
 if (SUCCEEDED(pDoc2->get_links(&pElementCol)))
 {
 DWORD len=0;
 if (SUCCEEDED(pElementCol->get_length((long*)&len)))
 {
 if (len!=0)
 {
 m_Num = m_Num+len;
 UpdateData(FALSE);
 for (long i=0; i<=(len-1); i++)
 {
 CComBSTR String;
 variant t index = i;
 if (SUCCEEDED(pElementCol->item(index,
 index, &spDispatch)))
 {
 if (SUCCEEDED(spDispatch->QueryInterface(
 IID_IHTMLAnchorElement, (void **) &pLoct)))
 {
 pLoct->get_href(&String);
 memset(host, 0, MAX_COMPUTER);
 lstrcpy(host, bstr_t(String));
 //添加到列表
 m_List.InsertItem(i, host);
 m_List.SetCheck(i, TRUE);
 pLoct->get_hostname(&String);
 }
 }
 }
 }
 }
 }
 }
 }
 }
 }
 }
}
```



```
memset(host,0,MAX_COMPUTER);
strcpy(host,_bstr_t(String));
if(strlen(host))
{
 m_List.SetItemText(i,1,host);
}
```

### 举一反三

根据本实例，读者可以：

- 提取网页源码。

## 13.7 其 他

电话应用程序接口（TAPI）是一种标准的应用程序接口（API），它可以使个人电脑通过运行微软公司的 Windows 系统来使用电话服务。电话应用程序接口（TAPI）被设计用来在同一个网络中同时支持 IP 及传统电话。本节通过实例介绍利用 TAPI 实现网络拨号的技术。

### 实例 385 利用 TAPI 实现网络拨号

这是一个自娱自乐的实例

实例位置：光盘\mingrisoft\13\385

### 实例说明

本实例通过 TAPI2.0 实现网络拨号。运行程序，在程序的列表中显示出支持 TAPI 的线路，选择支持拨号的线路，单击“拨号”按钮进行拨号，程序运行结果如图 13.18 所示。

### 技术要点

利用 TAPI 实现拨号，首先需通过 `lineInitializeEx` 函数对 `HLINEAPP` 对象进行初始化；然后通过 `lineGetDevCaps` 函数获得线路设备的相关信息；通过 `lineOpen` 函数打开进行拨号的线路；最后通过 `lineMakeCall` 函数进行拨号；通过 `lineDrop` 函数进行挂断。

(1) `lineInitializeEx` 函数。实例中主要通过 `lineInitializeEx` 函数初始化一个 `HLINEAPP` 对象，语法如下：

```
LONG lineInitializeEx(LPHLINEAPP lphLineApp,HINSTANCE hInstance,
LINECALLBACK lpfnCallback,LPCSTR lpszFriendlyAppName,LPDWORD lpdwNumDevs,
LPDWORD lpdwAPIVersion,LPLINEINITIALIZEEXPARAMS lpLineInitializeExParams);
```

参数说明：

- `lphLineApp`：将要初始化的 `HLINEAPP` 对象指针。
- `hInstance`：实例句柄。
- `lpfnCallback`：初始化过程中调用回调函数。
- `lpszFriendlyAppName`：`LINECALLINFO` 结构中指定的字符。
- `lpdwNumDevs`：设备数量。
- `lpdwAPIVersion`：函数的版本。
- `lpLineInitializeExParams`：线路的扩展参数。

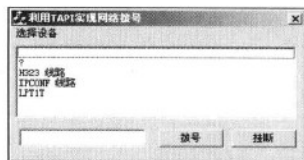


图 13.18 利用 TAPI 实现网络拨号

(2) lineGetDevCaps 函数。lineGetDevCaps 函数可以获得线路的相关信息，语法如下：

```
LONG lineGetDevCaps(HLINEAPP hLineApp,DWORD dwDeviceID,
 DWORD dwAPIVersion,DWORD dwExtVersion,LPLINEDEVCAPS lpLineDevCaps);
```

参数说明：

- hLineApp: HLINEAPP 对象指针。
- dwDeviceID: 线路设备的 ID 值。
- dwAPIVersion: 函数的版本值。
- dwExtVersion: 扩展的版本值。
- lpLineDevCaps: LINEDEVCAPS 结构指针，用来存储设备信息。

## 实现过程

(1) 新建名为 TAPITEL 的对话框 MFC 工程。

(2) 在对话框上添加列表控件，设置 ID 属性为 IDC\_LINELIST；添加文本编辑控件，设置 ID 属性为 IDC\_EDTELNUM；添加两个按钮控件，设置 ID 属性分别为 IDC\_BTIDIAL 和 IDC\_BTDROP，设置 Caption 属性分别为“拨号”和“挂断”。

(3) 在工程中添加 tapi32.lib 库。

(4) 在 StdAfx.h 文件中添加头文件引用：

```
ITAPI *gpTapi;
```

(5) 在 TAPITELDlg.h 文件中，自定义一个结构，存储呼叫的返回值，代码如下：

```
typedef struct _tagADDRESSINFO{
 HWND hStatus; //状态窗体句柄
 HWND hAnswer; //应答窗体句柄
 HCALL hCall; //呼叫句柄
 BOOL bCall; //是否可以呼叫
}ADDRESSINFO,*PADDRESSINFO;
```

(6) 在 TAPITELDlg.h 文件中定义变量，代码如下：

```
HANDLE ghCompletionPort;
PADDRESSINFO pAddressInfo;
HLINEAPP ghLineApp;
DWORD gdwAddresses;
DWORD gdwDeviceID;
HLINE ghLine;
HCALL call;
```

(7) 在 OnInitDialog 中列举出支持 TAPI 的所有线路，代码如下：

```
BOOL CTAPITELDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 LONG lResult;
 LINEINITIALIZEEXPARAMS exparams;
 DWORD dwDeviceID,dwNumDevs,dwAPIVersion, dwThreadId;
 exparams.dwTotalSize=sizeof(LINEINITIALIZEEXPARAMS);
 exparams.dwOptions=LINEINITIALIZEEXOPTION_USECOMPLETIONPORT;
 exparams.Handles.hCompletionPort=ghCompletionPort;
 lResult=lineInitializeEx(&ghLineApp,::AfxGetApp()->m_hInstance,NULL,"mrtapi",
 &dwNumDevs,&dwAPIVersion,&exparams);
 if (dwNumDevs==0)
 {lineShutdown(ghLineApp);
 return FALSE;}
 for(DWORD i=0;i<dwNumDevs;i++)
 {
 LINEDEVCAPS *pLineDevCaps;
 static DWORD dwMaxNeededSize=sizeof(LINEDEVCAPS);
 pLineDevCaps=(LINEDEVCAPS*)GlobalAlloc(GPTR,dwMaxNeededSize);
 pLineDevCaps->dwTotalSize=dwMaxNeededSize;
 lineGetDevCaps(ghLineApp,i,TAPI_CURRENT_VERSION,0,pLineDevCaps);
 dwMaxNeededSize=pLineDevCaps->dwNeededSize;
 pLineDevCaps=(LINEDEVCAPS*)GlobalReAlloc(HLOCAL)
 pLineDevCaps,dwMaxNeededSize,GMEM_MOVEABLE);
 //将可以拨号的设备添加到列表中
 if(pLineDevCaps->dwBearerModes==LINEBEARERMODE_VOICE)
 if(pLineDevCaps->dwMediaModes==LINEMEDIAMODE_INTERACTIVEVOICE)
 m_linelist.AddString(((LPCTSTR)(LPBYTE)
```

```
pLineDevCaps+pLineDevCaps->dwLineNameOffset));
}
return TRUE;
}
```

(8) 按钮“拨号”的实现函数，代码如下：

```
void CTAPITELDlg::OnDial()
{
 CString number;
 GetDlgItem(IDC_EDTELNUM)->GetWindowText(number);
 int i=m_linelist.GetCurSel();
 lineOpen(ghLineApp,i,&ghLine,TAPI_CURRENT_VERSION,
 0,0,LINECALLPRIVILEGE_OWNER,LINEMEDIAMODE_INTERACTIVEVOICE,NULL);
 ::lineMakeCall(ghLine,&call,number,0,0);
}
}
```

(9) 按钮“挂断”的实现函数，代码如下：

```
void CTAPITELDlg::OnDrop()
{
 lineDrop(call,NULL,0);
}
}
```

## 举一反三

根据本实例，读者可以：

- 利用 TAPI 实现对网络拨号的接听；
- 利用 TAPI 实现通信。

## 实例 386 互联网文件传输

这是一个自娱自乐的实例

实例位置：光盘\mingrisoft\13\386

## 实例说明

互联网文件传输与局域网文件传输不同，其特点主要是数据传输量大，经常出现发送的数据大小与接收的数据大小在一次发送或接收操作中不同的现象。这样的现象是由于在接收数据时可能接收了多个数据包或小于单个数据包。所以程序设计人员需要自己编程实现数据包接收的安全性和稳定性，程序运行结果如图 13.19 所示。

## 技术要点

为了解决数据发送与接收时数据大小的不同而导致的操作错误，可以在该实例中定义了一个名为 Cpackage 的类。该类可以看成是网络中数据发送与接收的最小单位（包），该类记录了发送或接收数据的信息。所以在数据接收时，以该类所提供的数据为依据对类中存储的数据进行处理就不会出现数据处理的错误。该类的定义如下：

```
//定义应用层数据包结构
//在文件开始发送时，数据缓冲区中前128个字节用于存储文件名，其后是文件数据
//而在文件发送过程中，数据缓冲区中均是文件数据
```

```
class CPackage
{
public:
 PackageType m_Type; //数据包类型
 SendFlag m_Flag; //文件发送标记
 DWORD m_dwSize; //数据报结构大小
 DWORD m_dwFileSize; //整个文件大小
 DWORD m_dwData; //m_Data的大小
 BYTE m_Data[]; //数据缓冲区
};
```

在发送数据时，首先根据发送的数据创建 CPackage 的对象，并将数据存储在类的数据缓

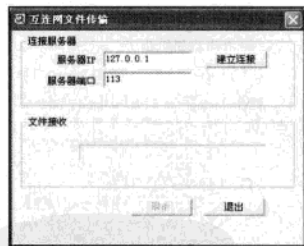


图 13.19 互联网文件传输

缓冲区 m\_Data 中。为了使数据具有安全性可以定义全局的临时缓冲区进行操作。实现代码如下：

```
//分包发送文件
//获取文件长度
CFile file;
file.Open(pDlg->m_szFileName, CFile::modeRead);
DWORD dwLen = file.GetLength() + 128; //128表示文件名占用的空间
//定义每次发送数据的大小
int nPerSendSize = 1024*6;
//计算需要分多少个数据包发送文件
int nPackageCount = dwLen / nPerSendSize;
int nMod = dwLen % nPerSendSize;

//发送第一个数据包，让对方确认是否接收文件
//确定第一个数据包的大小
int nFirstPackSize = 0;
if (nPackageCount > 0)
{
 nFirstPackSize = nPerSendSize;
}
else
{
 nFirstPackSize = nMod;
}

int nPackageSize = sizeof(CPackage);
HGLOBAL hGlobal = GlobalAlloc(GHND, nFirstPackSize + nPackageSize);
BYTE* pBuffer = (BYTE*)GlobalLock(hGlobal);
CPackage* pPackage = (CPackage*)pBuffer;
pPackage->m_Type = PTFILE;
pPackage->m_Flag = SFBEGIN;
pPackage->m_dwFileSize = dwLen;
pPackage->m_dwSize = nPackageSize;
pPackage->m_dwData = nFirstPackSize;

BYTE* pTmp = pBuffer + nPackageSize;
//设置文件名
memset(pTmp, 0, 128);
CString szName = file.GetFileName();
memcpy(pTmp, szName.GetBuffer(0), szName.GetLength());
szName.ReleaseBuffer();
pTmp = pTmp + 128;
//复制文件数据
file.ReadHuge(pTmp, nFirstPackSize - 128);
pDlg->m_ClientSock.Send(pBuffer, nFirstPackSize + nPackageSize);
GlobalUnlock(hGlobal);
GlobalFree(hGlobal);
```

从上段代码可以看出，本次发送是将文件名和部份文件数据发送出去。当网络的另一方接收数据时是根据 CPackage 类中指定的数据大小读取数据的，如果接收的数据与 CPackage 类中记录的大小不同时，继续接收数据直到满足 CPackage 类中指定的大小。处理这个包，同时继续接收下一个包的数据。这样就相当于在数据接收端建立了一个以包为单位的临时缓存区，只有当本次操作的缓存区满才执行操作。数据接收端代码如下：

```
if (m_hGlobal != NULL) //当全局缓冲区中存在数据
{
 BYTE* pData = (BYTE*)GlobalLock(m_hGlobal);
 int GSize = GlobalSize(m_hGlobal); //获取数据大小
 SunSize = GSize + nFact; //全局缓冲区数据与接收数据大小
 temp = new BYTE[SunSize]; //定义本次处理的临时缓冲区
 Ptemp = temp;
 memcpy(temp, pData, GSize); //复制全局缓冲区中数据
 memcpy(temp + GSize, pBuffer, nFact); //复制本次接收的数据
 GlobalUnlock(m_hGlobal);
 GlobalFree(m_hGlobal);
 m_hGlobal = NULL;
}
else
{
 SunSize = nFact;
 Ptemp = pBuffer;
}

CPackage* pPackage = (CPackage*)Ptemp; //获取需要处理的第一个包
while ((SunSize >= nPackage) && (SunSize >= pPackage->m_dwData))
{
 HandleRecData(pPackage); //处理包
```



```

SunSize -= nPackage + pPackage->m_dwData; //总大小减去已处理包大小
Ptemp += nPackage + pPackage->m_dwData; //将指针指向下一个包的地址
pPackage = (CPackage*)Ptemp; //获取本次处理的包
}
if (SunSize > 0) //将剩余数据写入全局缓冲区
{
 m_hGlobal = GlobalAlloc(GHND, SunSize);
 BYTE* pData = (BYTE*)GlobalLock(m_hGlobal);
 memcpy(pData, Ptemp, SunSize);
 GlobalUnlock(m_hGlobal);
}
else
{
 GlobalUnlock(m_hGlobal);
 GlobalFree(m_hGlobal);
 m_hGlobal = NULL;
}
}

```

## 实现过程

- (1) 新建一个基于对话框的工程，建立服务器端应用程序。
- (2) 在对话框中添加编辑框控件，用于发送文件的路径，添加进度条控件用于显示文件下载的进度情况，添加一个按钮控件，用来选择需要下载的文件，添加一个 Caption 属性为“发送”的按钮。
- (3) 由于在文件发送的服务器端，所发送的文件可能很大，所以发送操作应由一个独立的线程执行。定义一个名为 ThreadProc 的函数，用于线程执行。该函数实现了文件的分包发送传输。实现代码如下：

```

//定义线程函数，实现文件的发送
DWORD __stdcall ThreadProc(LPVOID lpParameter)
{
 CServerDlg* pDlg = (CServerDlg*) lpParameter;

 //分包发送文件
 //获取文件长度
 CFile file;
 file.Open(pDlg->m_szFileName, CFile::modeRead);
 DWORD dwLen = file.GetLength() + 128; //128表示文件名占用的空间
 //定义每次发送数据的大小
 int nPerSendSize = 1024*6;
 //计算需要分多少个数据包发送文件
 int nPackageCount = dwLen / nPerSendSize;
 int nMod = dwLen % nPerSendSize;

 //发送第一个数据包，让对方确认是否接收文件
 //确定第一个数据包的大小
 int nFirstPackSize = 0;
 if (nPackageCount > 0)
 {
 nFirstPackSize = nPerSendSize;
 }
 else
 {
 nFirstPackSize = nMod;
 }

 int nPackageSize = sizeof(CPackage);
 HGLOBAL hGlobal = GlobalAlloc(GHND, nFirstPackSize + nPackageSize);
 BYTE* pBuffer = (BYTE*)GlobalLock(hGlobal);
 CPackage* pPackage = (CPackage*) pBuffer;
 pPackage->m_Type = PTFILE;
 pPackage->m_Flag = SFBEGIN;
 pPackage->m_dwFileSize = dwLen;
 pPackage->m_dwSize = nPackageSize;
 pPackage->m_dwData = nFirstPackSize;

 BYTE* pTmp = pBuffer + nPackageSize;
 //设置文件名
 memset(pTmp, 0, 128);
 CString szName = file.GetFileName();
 memcpy(pTmp, szName.GetBuffer(0), szName.GetLength());
 szName.ReleaseBuffer();
 pTmp = pTmp + 128;
 //复制文件数据
 file.ReadHuge(pTmp, nFirstPackSize - 128);
 pDlg->m_ClientSock.Send(pBuffer, nFirstPackSize + nPackageSize);
}

```

```

GlobalUnlock(hGlobal);
GlobalFree(hGlobal);

pDlg->m_Progress.SetRange32(0, dwLen);
pDlg->m_Progress.SetPos(nFirstPackSize - 128);

//判断文件发送是否完成
if (nFirstPackSize >= dwLen) //一个数据包就完成了文件的发送
{
 file.Close();
 pDlg->m_Progress.SetPos(0);
 pDlg->m_bSending = FALSE;
 return 0;
}

//等待对方收受发送任务或取消发送任务
pDlg->m_RequestInfo = RIUNKNOWN;
while (true)
{
 if (pDlg->m_RequestInfo != RIUNKNOWN)
 break;
}
if (pDlg->m_RequestInfo == RIDENY) //对方拒绝接收文件
{
 pDlg->m_Progress.SetPos(0);
 file.Close();
 pDlg->m_bSending = FALSE;
 return 0;
}
else if (pDlg->m_RequestInfo == RICANCEL) //对方取消接收文件
{
 pDlg->m_Progress.SetPos(0);
 file.Close();
 pDlg->m_bSending = FALSE;
 return 0;
}
else if (pDlg->m_RequestInfo == RIACCEPT) //对方同意发送文件
{
 //分包继续发送文件
 for(int i=1; i<nPackageCount; i++)
 {
 //在发送过程中判断对方是否取消发送
 if (pDlg->m_RequestInfo == RICANCEL)
 {
 pDlg->m_Progress.SetPos(0);
 file.Close();
 pDlg->m_bSending = FALSE;
 return 0;
 }

 hGlobal = GlobalAlloc(GHND, nPerSendSize + nPackageSize);

 BYTE* pBuffer = (BYTE*)GlobalLock(hGlobal);
 CPackage* pPackage = (CPackage*) pBuffer;
 pPackage->m_Type = PTFILE;
 if (nMod == 0 && i == nPackageCount-1)
 pPackage->m_Flag = SFEND;
 else
 pPackage->m_Flag = SFSENDING;

 pPackage->m_dwFileSize = dwLen;
 pPackage->m_dwSize = nPackageSize;
 pPackage->m_dwData = nPerSendSize;
 //设置文件名
 BYTE* pTmp = pBuffer + nPackageSize;
 //复制文件数据
 file.ReadHuge(pTmp, nPerSendSize);
 pDlg->m_ClientSock.Send(pBuffer, nPerSendSize + nPackageSize);
 int nPos = pDlg->m_Progress.GetPos();
 nPos += nPerSendSize;
 pDlg->m_Progress.SetPos(nPos);

 GlobalUnlock(hGlobal);
 GlobalFree(hGlobal);
 }
 if (nPackageCount > 0 && nMod != 0) //防止文件太小导致重新发送文件
 {
 //发送最后一次数据包
 hGlobal = GlobalAlloc(GHND, nMod + nPackageSize);
 }
}

```



```

 BYTE* pBuffer = (BYTE*)GlobalLock(hGlobal);
 CPackage* pPackage = (CPackage*) pBuffer;
 pPackage->m_Type = PTFILE;
 pPackage->m_Flag = SFEND;
 pPackage->m_dwFileSize = dwLen;
 pPackage->m_dwSize = nPackageSize;
 pPackage->m_dwData = nMod;
 //设置文件名
 BYTE* pTmp = pBuffer + nPackageSize;
 //复制文件数据
 file.ReadHuge(pTmp, nMod);
 pDlg->m_ClientSock.Send(pBuffer, nMod + nPackageSize);
 GlobalUnlock(hGlobal);
 GlobalFree(hGlobal);
 }
 pDlg->m_Progress.SetPos(0);
 file.Close();
 return 0;
}

```

(4) 在窗体中单击“发送”按钮，实现线程的创建与文件发送。实现代码如下：

```

//发送文件
void CServerDlg::OnSendFile()
{
 if (m_bSending)
 {
 MessageBox("当前文件发送没有完整，不能发送新文件");
 return;
 }
 //判断文件是否存在
 CString szFileName;
 m_FileName.GetWindowText(szFileName);

 if (!szFileName.IsEmpty())
 {
 CFileFind flFind;
 BOOL bRet = flFind.FindFile(szFileName);
 if (bRet)
 {
 //创建一个线程实现发送文件
 m_szFileName = szFileName;
 m_hSendThread = CreateThread(NULL, 0, ThreadProc, this, 0, 0);
 }
 else
 {
 MessageBox("文件不存在!");
 }
 }
}

```

(5) 在窗体类中实现 ReceiveData 方法用于接收客户端的数据，获取客户端反馈的命令。实现代码如下：

```

//接收数据
void CServerDlg::ReceiveData()
{
 //接收对方发来的应答信息
 int nPackageSize = sizeof(CPackage);
 BYTE* pBuffer = new BYTE[nPackageSize];
 int nFact = m_ClientSock.Receive(pBuffer, nPackageSize);
 if (nFact >= nPackageSize)
 {
 CPackage* pPackage = (CPackage*) pBuffer;
 if (pPackage->m_Flag == SFCANCEL) //对方取消文件接收
 {
 m_RequestInfo = RICANCEL;
 }
 else if (pPackage->m_Flag == SFDENY) //对方拒绝接收文件
 {
 m_RequestInfo = RIDENY;
 }
 else if (pPackage->m_Flag == SFACCEPT) //对方同意接收文件
 {
 m_RequestInfo = RIACCEPT;
 }
 }
 delete [] pBuffer;
}

```



(6) 新建一个基于对话框的应用程序，创建客户端应用程序。

(7) 在窗体上添加两个编辑框控件，用于指定服务器端的 IP 地址和 Port 端口号。添加 1 个进度条控件用来显示当前文件下载的进度情况。添加 3 个按钮控件，分别设置其 Caption 属性为“建立连接”、“取消”和“退出”。

(8) 在窗体类中实现 HandleRecData 方法，该方法用于处理以包为单位的数据，并将数据写入下载的文件中。实现代码如下：

```
void CClientDlg::HandleRecData(CPackage *pPackage)
{
 if (pPackage != NULL)
 {
 if (pPackage->m_Type==PTFILE) //只处理文件数据
 {
 static CFile file;

 if (pPackage->m_Flag==SFBEGIN) //开始发送
 {
 //读取文件名
 char szFileName[128] = {0};
 memcpy(szFileName, pPackage->m_Data, 128);
 CFileDialog fDlg(FALSE, "", szFileName);
 if (fDlg.DoModal()==IDOK)
 {
 if (m_bSending) //当前文件发送没有结束，又发送一个文件
 {
 file.Close();
 }

 m_Progress.SetPos(0);
 //确定整个文件大小
 DWORD dwFileSize = pPackage->m_dwFileSize;
 //设置进度条的表示范围
 m_Progress.SetRange32(0, dwFileSize);

 CString szFile = fDlg.GetPathName();
 file.Open(szFile,
 CFile::modeCreate|CFile::modeReadWrite);
 //计算数据报中文件数据的大小
 int nFileLen = pPackage->m_dwData - 128;
 //文件比较小，只发送一次数据包就完成了文件发送
 if (nFileLen >= dwFileSize)
 {
 m_bSending = FALSE; //设置结束标记
 file.Close();
 m_Progress.SetPos(0);
 m_Cancel.EnableWindow(FALSE);
 return;
 }

 //读取数据包中的文件数据
 BYTE* pTmp = pPackage->m_Data;
 pTmp += 128; //略过文件名数据，定位到文件数据
 file.WriteHuge(pTmp, nFileLen);
 m_Progress.SetPos(nFileLen);
 //发送接收标记
 CPackage package;
 package.m_Type = PTFILE;
 package.m_Flag = SFACCEPT;
 package.m_dwSize = sizeof(CPackage);
 package.m_dwData = 0;
 package.m_dwFileSize = 0;
 m_ClientSock.Send(&package, sizeof(CPackage));

 m_bSending = TRUE;
 m_Cancel.EnableWindow();
 }
 }
 else
 {
 //取消文件发送
 CPackage package;
 package.m_Type = PTFILE;
 package.m_Flag = SFDENY;
 package.m_dwSize = sizeof(CPackage);
 package.m_dwData = 0;
 }
 }
 }
}
```



```

package.m_dwFileSize = 0;
m_ClientSock.Send(&package, sizeof(CPackage));
m_Cancel.EnableWindow();

}

}
else if (pPackage->m_Flag==SFSENDING) //文件发送中
{
 file.WriteHuge(pPackage->m_Data, pPackage->m_dwData);
 int nPos = m_Progress.GetPos();
 nPos += pPackage->m_dwData;
 m_Progress.SetPos(nPos);
}
else if (pPackage->m_Flag==SFEND) //文件发送结束
{
 file.WriteHuge(pPackage->m_Data, pPackage->m_dwData);
 m_bSending = FALSE; //设置结束标记
 m_Cancel.EnableWindow(FALSE);
 file.Close();
 m_Progress.SetPos(0);
}
else if (pPackage->m_Flag==SFCANCEL) //对方取消了文件发送
{
 m_bSending = FALSE; //设置结束标记
 file.Close();
 m_Cancel.EnableWindow(FALSE);
 m_Progress.SetPos(0);
 MessageBox("对方取消了文件发送!");
}
}
}
}
}

```

(9) 在窗体类中实现 ReceiveData 方法, 该方法用于接收服务器端发来的数据, 并将数据组合或分解成若干个包进行处理。实现代码如下:

```

//接收数据
void CClientDlg::ReceiveData()
{
 static int nMaxLen = 1024*100; //定义接收最大长度
 HGLOBAL hGlobal = GlobalAlloc(GHND, nMaxLen); //定义接收全局缓冲区
 BYTE* pBuffer = (BYTE*)GlobalLock(hGlobal); //获取指定全局缓冲区指针
 int nFact = m_ClientSock.Receive(pBuffer, nMaxLen); //接收数据
 int nPackage = sizeof(CPackage); //计算数据包大小
 int SunSize = 0; //本次处理数据总大小
 BYTE *Ptemp = NULL; //处理缓冲区数据指针
 BYTE *temp = NULL; //全局缓冲区指针
 if (m_hGlobal != NULL) //当全局缓冲区中存在数据
 {
 BYTE* pData = (BYTE*)GlobalLock(m_hGlobal);
 int GSize = GlobalSize(m_hGlobal); //获取数据大小
 SunSize = GSize + nFact; //全局缓冲区数据与接收数据大小
 temp = new BYTE[SunSize]; //定义本次处理的临时缓冲区
 Ptemp = temp;
 memcpy(temp, pData, GSize); //复制全局缓冲区中数据
 memcpy(temp + GSize, pBuffer, nFact); //复制本次接收的数据
 GlobalUnlock(m_hGlobal);
 GlobalFree(m_hGlobal);
 m_hGlobal = NULL;
 }
 else
 {
 SunSize = nFact;
 Ptemp = pBuffer;
 }

 CPackage* pPackage = (CPackage*)Ptemp; //获取需要处理的第一个包
 while ((SunSize >= nPackage) && (SunSize >= pPackage->m_dwData))
 {
 HandleRecData(pPackage); //处理包
 SunSize -= nPackage + pPackage->m_dwData; //总大小减去已处理包大小
 Ptemp += nPackage + pPackage->m_dwData; //将指针指向下一个包的地址
 pPackage = (CPackage*)Ptemp; //获取本次处理的包
 }
 if (SunSize > 0) //将剩余数据写入全局缓冲区
 {
 m_hGlobal = GlobalAlloc(GHND, SunSize);
 BYTE* pData = (BYTE*)GlobalLock(m_hGlobal);
 }
}

```

```
 memcpy(pData,Ptemp,SunSize);
 GlobalUnlock(m_hGlobal);
 }
 else
 {
 GlobalUnlock(m_hGlobal);
 GlobalFree(m_hGlobal);
 m_hGlobal = NULL;
 }

 if (temp != NULL)
 delete temp;

 GlobalUnlock(hGlobal);
 GlobalFree(hGlobal);
}
```

### 举一反三

根据本实例，读者可以：

- 利用 TAPI 实现对网络拨号的接听；
- 利用 TAPI 实现通信。



## 第 14 章

# 加密、安全与软件注册

- 数据加密与解密
- 软件注册与加密

Visual C++



## 14.1 数据加密与解密

在商业企业的计算机中往往存在大量的机密文件, 这些机密文件对企业的发展将会产生不可估量的作用。如果这些机密文件保管不善, 将会使企业遭受巨大的损失。下面几个对数据或文件进行加密处理的实例, 将会有效地防止机密文件被盗取或查阅的情况发生。

### 实例 387 数据加密技术

本实例是一个提高效率、人性化的程序

实例位置: 光盘\mingrisoft\14\387

#### 实例说明

在一些应用程序或网络程序中, 经常会存有一些非常机密的文件或数据, 为了防止其他非法用户查阅或盗取这些机密数据, 可对其进行加密。运行程序, 在“密钥”编辑框中输入密钥, 在“待加密的字符串”编辑框中输入要加密的字符串, 单击“加密”按钮, 密文将显示在“加密后的字符串”编辑框中, 如图 14.1 所示。

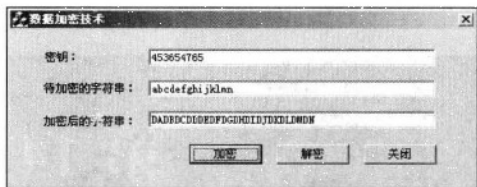


图 14.1 数据加密技术

#### 技术要点

通过使用 GetAt 和 SetAt 函数可以将密文与密钥提取出来的字符组成新的 ASCII 字符, 从而实现加密。下面介绍这两个函数。

GetAt 函数: 返回字符串内指定的单个字符, 语法如下:

```
TCHAR GetAt(int nIndex) const;
```

参数说明:

- nIndex: 是返回字符在字符串的位置。
- 返回值: 字符串中的单个字符。
- SetAt 函数: 在字符串的指定位置写入单个字符, 语法如下:

```
void SetAt(int nIndex, TCHAR ch);
```

参数说明:

- nIndex: 插入字符的位置。
- ch: 要插入的字符。

#### 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“数据加密技术”。
- (2) 在窗体上添加 3 个文本编辑控件和 3 个按钮控件。
- (3) 加密代码如下:

```
CString CDataencryptDlg::Encrypt(CString S, WORD K)
{
 CString Str, Str1, Result;
 int i, j;
 Str = S;
 for(i=0; i<S.GetLength(); i++) //根据字符串长度循环
 {
 Str.SetAt(i, S.GetAt(i)+K); //获得字符
 }
 S = Str;
 //加密字符
}
```

```

for(i=0;i<S.GetLength();i++)
{
 j = (BYTE)S.GetAt(i);
 Str1 = "01";
 Str1.SetAt(0,65+j/26);
 Str1.SetAt(1,65+j%26);
 Result += Str1;
}
return Result;
}

```

#### (4) 解密代码如下:

```

CString CDataencryptDlg::Decrypt(CString S, WORD K)
{
 CString Result,Str;
 int i,j;
 //解密字符
 for(i=0;i<S.GetLength()/2;i++)
 {
 j=((BYTE)S.GetAt(2*i)-65)*26;
 j+=(BYTE)S.GetAt(2*i+1)-65;
 Str = "0";
 Str.SetAt(0,j);
 Result += Str;
 }
 S = Result;
 for(i=0;i<S.GetLength();i++)
 {
 Result.SetAt(i,(BYTE)S.GetAt(i)-K); //设置字符串
 }
 return Result;
}

```

### 举一反三

根据本实例，读者可以：

- 在系统的口令表中进行口令加密，防止他人查看明文，进入系统。

## 实例 388

### 使用 MD5 算法对密码进行加密

本实例是一个提高基础技能的程序

实例位置：光盘\mingrisoft\14\388

### 实例说明

密码加密是软件系统所应具备的基本功能，许多程序设计人员为了方便只是将用户设定的密码以明文的形式存入数据库或文件中。这样做非常严重的危害了系统运行的安全性，用户设置的密码很容易就会被发现。所以为了增加系统的安全性，最好对用户设置的密码进行加密，这样别人即使看到了密码也只是加密后的密码。该实例使用的是 MD5 算法，该算法是一个不能反向解密的算法，如图 14.2 所示。

### 技术要点

许多程序设计人员认为 MD5 算法是一个非常复杂的算法，所以没有自己去实现，其实只要按照 MD5 算法的原理设计程序并不是一件复杂的事。MD5 算法以 512 位分组来处理输入的信息，且每一分组又被划分为 16 个 32 位子分组，经过了一系列的处理后，算法的输出由 4 个 32 位分组组成，将这 4 个 32 位分组级联后将生成一个 128 位散列值（16 个字符）。

在 MD5 中首先定义 4 个 32 位的 16 进制数，被称为链接变量。定义代码如下：

```

#define MD5_INIT_STATE_0 0x67452301
#define MD5_INIT_STATE_1 0xefcdab89
#define MD5_INIT_STATE_2 0x98badcfe
#define MD5_INIT_STATE_3 0x10325476

```

当 4 个链接变量设置完成后就可以对输入的数据进行 4 轮主循环了，而每轮主循环又进行

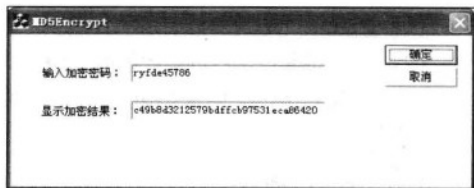


图 14.2 使用 MD5 算法对密码进行加密

16 次操作。每轮循环对应了一个操作函数，通过参数的不同返回 4 个 32 位的变量值，当所有循环结束后再将最终得到的 4 个 32 位变量值与链接变量相加赋给对应的链接变量。这 4 个操作函数的算法如下：

```
FF(X,Y,Z)=(X&Y)((~X)&Z) //第1轮调用
GG(X,Y,Z)=(X&Z)((Y&(~Z)) //第2轮调用
HH(X,Y,Z)=X^Y^Z //第3轮调用
II(X,Y,Z)=Y^X((~Z)) //第4轮调用
```

在上面的算法中 X、Y 和 Z 的取值分别是用户输入数据、MD5\_S 数组和 MD5\_T 数组中取得，其中 X 值是根据 XINDEX 数组中的值作为索引获取的。XINDEX 数组和 MD5\_S 数组中的值可由程序员自行修改来改变 MD5 加密的结果。而 MD5\_T 数组中的数据是固定由 4294967296(2 的 32 次方)\*abs(fsin(索引值))计算得来的。这 3 个数组的定义分别如下：

```
//参加运算的S盒
static DWORD MD5_S[4][16] =
{
 {7,12,17,22,9,6,8,5,7,4,17,3,7,2,17,10},
 {5,9,14,20,7,4,10,12,5,8,14,6,5,9,3,20},
 {4,11,16,23,5,8,10,12,4,7,9,23,8,11,22,14},
 {6,10,15,21,6,10,15,21,6,10,15,21,6,10,15,21}
};
//获取加密数据的索引表
static int XINDEX[4][16] =
{
 {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15},
 {1,6,11,0,5,10,15,4,9,14,3,8,13,2,7,12},
 {5,8,11,14,1,4,7,10,13,0,3,6,9,12,15,2},
 {0,7,14,5,12,3,10,1,8,15,6,13,4,11,2,9}
};
//参加运算的数组
static DWORD MD5_T[64] =
{
 0xd76aa478,0xe8c7b756,0x242070db,0xc1bdccee,
 0xf57c0faf,0x4787c62a,0xa8304613,0xfd469501,
 0x698098d8,0x8b44f7af,0xffff5bb1,0x895cd7be,
 0x6b901122,0xfd987193,0xa679438e,0x49b40821,

 0xf61e2562,0xc040b340,0x265e5a51,0xe9b6c7aa,
 0xd62f105d,0x02441453,0xd8a1e681,0xe7d3fbc8,
 0x21e1cde6,0xc33707d6,0xf4d50d87,0x455a14ed,
 0xa9e3e905,0xfcefa3f8,0x676f02d9,0x8d2a4c8a,

 0xfffa3942,0x8771f681,0x6d9d6122,0xfde5380c,
 0xa4beea44,0x4bdecfa9,0xf6bb4b60,0xbee5fb60,
 0x289b7ec6,0xeaad127fa,0xd4ef3085,0x04881d05,
 0xd9d4d039,0xe6db99e5,0x1fa27cf8,0xc4ac5665,

 0xf4292244,0x432aff97,0xab9423a7,0xfc93a039,
 0x655b59c3,0x8f0ccc92,0xffeff74d,0x85845dd1,
 0x6fa87e4f,0xfe2ce6e0,0xa3014314,0x4e0811a1,
 0xf7537e82,0xbd3af235,0x2ad7d2bb,0xeb86d391
};
```

## 实现过程

- (1) 新建一个基于对话框的工程。
- (2) 在对话框上添加两个文本编辑框控件和两个按钮控件。
- (3) 首先对 MD5 加密类进行定义。实现代码如下：

```
class CMD5Class
{
private:
 BYTE m_lpszBuffer[64]; //存储加密数据的缓冲区
 ULONG m_nCount[2]; //存储预定义的链接变量
 ULONG m_IMD5[4];

 void Transform(BYTE Block[64]); //对64位数据进行4轮加密
 void Update(BYTE* Input, ULONG nInputLen); //更新缓冲区执行加密
 CString Final(); //生成加密结果

 inline DWORD RotatelLeft(DWORD x, int n);
 inline void FF(DWORD& A, DWORD B, DWORD C, DWORD D, DWORD X, DWORD S, DWORD T);
 inline void GG(DWORD& A, DWORD B, DWORD C, DWORD D, DWORD X, DWORD S, DWORD T);
 inline void HH(DWORD& A, DWORD B, DWORD C, DWORD D, DWORD X, DWORD S, DWORD T);
 inline void II(DWORD& A, DWORD B, DWORD C, DWORD D, DWORD X, DWORD S, DWORD T);

 void DWordToByte(BYTE* Output, DWORD* Input, UINT nLength); //整型转比特
 void ByteToDWord(DWORD* Output, BYTE* Input, UINT nLength); //比特转整型
};
```

```
public:
CMD5Class();
virtual ~CMD5Class() {};
static CString GetMD5(BYTE* pBuf, UINT nLength); //数据加密
};
```

(4) 在 MD5 类的构造函数中对链接变量进行赋值。实现代码如下:

```
CMD5Class::CMD5Class()
{
 memset(m_lpszBuffer, 0, 64);
 m_nCount[0] = m_nCount[1] = 0;

 m_IMD5[0] = MD5_INIT_STATE_0;
 m_IMD5[1] = MD5_INIT_STATE_1;
 m_IMD5[2] = MD5_INIT_STATE_2;
 m_IMD5[3] = MD5_INIT_STATE_3;
}
```

(5) 创建比特值与整型的互转函数, 实现代码如下:

```
void CMD5Class::ByteToDWord(DWORD* Output, BYTE* Input, UINT nLength)
{
 ASSERT(nLength % 4 == 0);
 ASSERT(AfxIsValidAddress(Output, nLength/4, TRUE));
 ASSERT(AfxIsValidAddress(Input, nLength, FALSE));

 UINT i=0;
 UINT j=0;

 for (; j < nLength; i++, j += 4) //每4个比特值转成一个整型值
 {
 Output[i] = (ULONG)Input[j] |
 (ULONG)Input[j+1] << 8 |
 (ULONG)Input[j+2] << 16 |
 (ULONG)Input[j+3] << 24;
 }
}

void CMD5Class::DWordToByte(BYTE* Output, DWORD* Input, UINT nLength)
{
 ASSERT(nLength % 4 == 0);
 ASSERT(AfxIsValidAddress(Output, nLength, TRUE));
 ASSERT(AfxIsValidAddress(Input, nLength/4, FALSE));

 UINT i = 0;
 UINT j = 0;
 for (; j < nLength; i++, j += 4) //每个整型值转成4个比特值
 {
 Output[j] = (UCHAR)(Input[i] & 0xff);
 Output[j+1] = (UCHAR)((Input[i] >> 8) & 0xff);
 Output[j+2] = (UCHAR)((Input[i] >> 16) & 0xff);
 Output[j+3] = (UCHAR)((Input[i] >> 24) & 0xff);
 }
}
```

(6) 实现 4 轮主循环中, 每轮循环所调用的计算方法。实现代码如下:

```
void CMD5Class::FF(DWORD& A, DWORD B, DWORD C, DWORD D, DWORD X, DWORD S, DWORD T)
{
 DWORD F = (B & C) | (~B & D);
 A += F + X + T;
 A = RotateLeft(A, S);
 A += B;
}

void CMD5Class::GG(DWORD& A, DWORD B, DWORD C, DWORD D, DWORD X, DWORD S, DWORD T)
{
 DWORD G = (B & D) | (C & ~D);
 A += G + X + T;
 A = RotateLeft(A, S);
 A += B;
}

void CMD5Class::HH(DWORD& A, DWORD B, DWORD C, DWORD D, DWORD X, DWORD S, DWORD T)
{
 DWORD H = (B ^ C ^ D);
 A += H + X + T;
 A = RotateLeft(A, S);
 A += B;
}

void CMD5Class::II(DWORD& A, DWORD B, DWORD C, DWORD D, DWORD X, DWORD S, DWORD T)
{
 DWORD I = (C ^ (B | ~D));
 A += I + X + T;
```



```
A = RotateLeft(A, S);
A += B;
```

(7) 在 MD5 类中实现 Transform 方法用于执行 4 轮主循环对数据进行加密转换, 并获得加密后的链接变量。实现代码如下:

```
void CMD5Class::Transform(BYTE Block[64])
{
 ULONG a = m_lMD5[0]; //链接变量
 ULONG b = m_lMD5[1];
 ULONG c = m_lMD5[2];
 ULONG d = m_lMD5[3];

 ULONG X[16];
 ByteToDWord(X, Block, 64); //将64个比特值转成16个整型值
 for (int i=0;i<4;i++) //4轮主循环
 {
 for (int j=0;j<16;j++) //16个子循环
 {
 switch (i)
 {
 case 0:
 FF(a,b,c,d,X[XINDEX[i][j]],MD5_S[i][j],MD5_T[(i+1)*(j+1)]);
 break;
 case 1:
 GG(a,b,c,d,X[XINDEX[i][j]],MD5_S[i][j],MD5_T[(i+1)*(j+1)]);
 break;
 case 2:
 HH(a,b,c,d,X[XINDEX[i][j]],MD5_S[i][j],MD5_T[(i+1)*(j+1)]);
 break;
 case 3:
 II(a,b,c,d,X[XINDEX[i][j]],MD5_S[i][j],MD5_T[(i+1)*(j+1)]);
 break;
 }
 }
 m_lMD5[0] += a;
 m_lMD5[1] += b;
 m_lMD5[2] += c;
 m_lMD5[3] += d;
 }
}
```

(8) 在类中实现 Update 方法用于处理加密数据。实现代码如下:

```
void CMD5Class::Update(BYTE* Input, ULONG nInputLen)
{
 UINT nIndex = (UINT)((m_nCount[0] >> 3) & 0x3F);

 if ((m_nCount[0] += nInputLen << 3) < (nInputLen << 3))
 {
 m_nCount[1]++;
 }
 m_nCount[1] += (nInputLen >> 29);

 UINT i=0;
 UINT nPartLen = 64 - nIndex;
 if (nInputLen >= nPartLen) //数据是否大于等于64位
 {
 //复制64位数据到缓冲区
 memcpy(&m_lpszBuffer[nIndex], Input, nPartLen);
 Transform(m_lpszBuffer); //加密
 for (i = nPartLen; i + 63 < nInputLen; i += 64) //加密剩余数据
 {
 Transform(&Input[i]);
 }
 nIndex = 0;
 }
 else
 {
 i = 0;
 }
 //将未加密的剩余数据写入缓冲区
 memcpy(&m_lpszBuffer[nIndex], &Input[i], nInputLen-i);
}
```

(9) 实现 Final 方法完成对数据的加密, 并将链接变量转换成 16 个长度的字符串进行输出。实现代码如下:

```
CString CMD5Class::Final()
{
 BYTE Bits[8];
 DWordToByte(Bits, m_nCount, 8);
 UINT nIndex = (UINT)((m_nCount[0] >> 3) & 0x3f);
 //56是64-8;120是64*2-8
```

```
UINT nPadLen = (nIndex < 56) ? (56 - nIndex) : (120 - nIndex);
Update(PADDING, nPadLen);

Update(Bits, 8);
const int nMD5Size = 16;
unsigned char lpszMD5[nMD5Size];
DWordToByte(lpszMD5, m_lMD5, nMD5Size); //将链接变量转成比特值

CString strMD5;
for (int i=0; i < nMD5Size; i++) //生成十六进制表示的加密后数据
{
 CString Str;
 if (lpszMD5[i] == 0) {
 Str = CString("00");
 }
 else if (lpszMD5[i] <= 15) {
 Str.Format("0%x", lpszMD5[i]);
 }
 else {
 Str.Format("%x", lpszMD5[i]);
 }

 ASSERT(Str.GetLength() == 2);
 strMD5 += Str;
}
ASSERT(strMD5.GetLength() == 32);
return strMD5;
}
```

(10) 在窗体上单击“确定”按钮对数据进行加密。实现代码如下:

```
void CMD5EncryptDlg::OnEncrypt()
{
 UpdateData();
 CString str = MD.GetMD5((BYTE *)m_password.GetBuffer(0),
 m_password.GetLength());
 m_edit = str;
 UpdateData(false);
}
```

### 举一反三

根据本实例,读者可以:

- 对网络中传输的文件进行加密与解密。

## 实例 389

### 对数据包进行加密保障通信安全

本实例可以提高计算机安全性

实例位置: 光盘\mingrisoft\14\389

### 实例说明

现在网络方面的应用程序非常多,不管是聊天软件、游戏还是管理软件,都可以通过在网络中发送数据来实现不同地区的数据交换。但有许多网络软件的开发者和生产商却对他们的产品缺少安全意识,没有对网络中传送的数据包进行加密,使得一些不法分子可以通过截获数据包来进行伪造获取利益。该实例所实现的聊天程序在发送数据时,对数据进行了加密,当接收数据时对数据进行解密,这样就可以提高数据在网络中传播的安全性,如图 14.3 所示。

### 技术要点

对网络数据包进行加密也就是对发送的数据进行加密。而对发送的数据进行加密/解密有许多种算法,比如 DES 加密/解密算法和 AES 加密/解密算法等。在该实例中笔者实现了通过一个简单的加密/解密算法来对需要发送的网络数据进行加密操作。加密函数的实现代码如下:

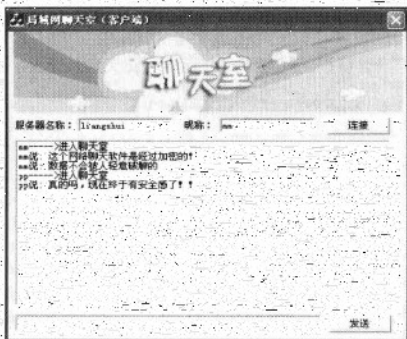


图 14.3 对数据包进行加密保障通信安全

```
CString CClientDlg::Encrypt(CString s,WORD k)
```

```
{
 CString Str,Str1,Result;
 int i,j;
 Str = s;
 for(i=0;i<s.GetLength();i++)
 {
 Str.SetAt(i,s.GetAt(i)+k);
 }
 s = Str;
 for(i=0;i<s.GetLength();i++)
 {
 j = (BYTE)s.GetAt(i);
 Str1 = "01";
 Str1.SetAt(0,65+j/26);
 Str1.SetAt(1,65+j%26);
 Result += Str1;
 }
 return Result;
}
```

解密函数与加密函数是对应的反向操作。实现代码如下：

```
CString CClientDlg::Decrypt(CString s,WORD k)
```

```
{
 CString Result,Str;
 int i,j;
 for(i=0;i<s.GetLength()/2;i++)
 {
 j=((BYTE)s.GetAt(2*i)-65)*26;
 j+=(BYTE)s.GetAt(2*i+1)-65;
 Str = "0";
 Str.SetAt(0,j);
 Result += Str;
 }
 s = Result;
 for(i=0;i<s.GetLength();i++)
 {
 Result.SetAt(i,(BYTE)s.GetAt(i)-k);
 }
 return Result;
}
```

## 实现过程

- (1) 新建一个基于对话框的工程 (服务器端)。
- (2) 在窗体类中实现 AcceptConnect 方法连接客户端。实现代码如下：

```
void CServerDlg::AcceptConnect()
{
 CClientSocket* socket = new CClientSocket(this); //创建客户端套接字
 //接受客户端的连接
 if (m_pSocket->Accept(*socket))
 m_socketlist.AddTail(socket); //添加到客户端套接字列表
 else
 delete socket;
}
```

- (3) 在窗体类中实现 ReceiveData 方法用于对接收的客户端数据进行转发,在这个方法中由于服务器只是起到转发作用,所以不用对数据进行解密。实现代码如下：

```
void CServerDlg::ReceiveData(CClientSocket* socket)
{
 char bufferdata[BUFFERSIZE]; //定义接收数据缓冲区
 //接收客户端传来的数据
 int result = socket->Receive(bufferdata,BUFFERSIZE);
 bufferdata[result] = 0; //在末端添加结束符
 POSITION pos = m_socketlist.GetHeadPosition();
 //将数据发送给每个客户端
 while (pos!=NULL)
 {
 CClientSocket* socket = (CClientSocket*)m_socketlist.GetNext(pos);
 if (socket != NULL)
 socket->Send(bufferdata,result); //向客户端发送数据
 }
}
```

- (4) 新建一个基于对话框的应用程序,将窗体标题改为“局域网聊天程序 (客户端)”。
- (5) 在窗体上添加 1 个图片控件、3 个文本编辑框控件、1 个列表框控件和两个按钮控件。
- (6) 当用户按下“连接”按钮时将连接客户端与服务器端,这时需要向服务器发送数据,

此时的数据就需要对其进行加密操作。实现代码如下：

```
void CClientDlg::OnButtonjoin()
{
 UpdateData(true);
 CString servername = m_servername; //读取服务器名称
 int port; //获取端口
 port = 70;
 if (!pMysocket->Connect(servername,port)) //连接服务器
 {
 MessageBox("连接服务器失败!");
 return;
 }
 CString str;
 str.Format("%s----->%s",m_name,"进入聊天室"); //格式化发送文本
 str = Enjcrypt(str,123456); //加密数据
 //向服务器发送连接数据
 int num = pMysocket->Send(str.GetBuffer(0),str.GetLength());
}
```

(7) 当在消息框中输入需要发送的文本信息后单击“发送”按钮时，消息文本将被发送到服务器端，此时的数据同样也需要加密。实现代码如下：

```
void CClientDlg::OnButtonsend()
{
 CString str,temp;
 m_info.GetWindowText(str); //获取消息文本
 if (str.IsEmpty()||m_name.IsEmpty()) //判断消息文本与用户名是否为空
 return;
 temp.Format("%s说: %s",m_name,str); //格式化消息文本
 temp = Enjcrypt(temp,123456); //加密数据
 //发送消息文本到服务器端
 int num = pMysocket->Send(temp.GetBuffer(temp.GetLength()),
 temp.GetLength());
 m_info.SetWindowText(""); //清空消息文本框
 m_info.SetFocus(); //指定输入焦点
}
```

(8) 添加 ReceiveData 成员方法，用于接收服务器传来的数据，此时需要对接收到的数据进行解密再显示在窗体中。代码如下：

```
void CClientDlg::ReceiveData()
{
 char buffer[200]; //指定接收数据的缓冲区
 //接收传来的数据
 int factdata = pMysocket->Receive(buffer,200);

 buffer[factdata] = '\0'; //设置结束符
 CString str;
 str.Format("%s",buffer); //转成字符串
 str = Decrypt(str,123456); //解密
 int i = m_list.GetCount();
 //将数据添加到列表框中
 m_list.InsertString(m_list.GetCount(),str);
}
```

### 举一反三

根据本实例，读者可以：

- 多报交错数据加密。

## 实例 390 对档案进行加密和解密

本实例可以提高计算机安全性

实例位置：光盘\mingrisoft\14\390

### 实例说明

加密/解密以成为人们工作与生活中的一部分，在工作中对于一些重要的文件需要加密，不允许其他人打开，在生活中个人的日记、照片等都属于个人隐私，也不希望别人看到，所以加密和解密已走进人们的生活。对档案进行加密也就是对文件进行加密，在该实例中所使用的加密/解密算法是 AES 算法。如图 14.4 所示。



## 技术要点

AES 是一个新的可以用于保护电子数据的加密算法。明确地说, AES 是一个迭代的、对称密钥分组的密码, 它可以使用 128、192 和 256 位密钥, 并且用 128 位 (16 字节) 分组加密和解密数据。

AES 算法是基于置换和代替的。置换是数据的重新排列, 而代替是用一个单元数据替换另一个。AES 使用了几种不同的技术来实现置换和替换。

在 AES 算法中定义了  $16 \times 16$  的二维数组 SBox 与 InvSBox, 这两个数组分别用来置换与反置换加密/解密的数据。数组 w 则是可改变长度的二维数组, 该数组做为密钥调度表将参与轮密钥加运算 (圈密钥加)。而密钥调度表的生成则是用户指定密钥与 Rc 数组运算得来的。这 3 个数组的定义如下:

```
byte SBox[16][16] =
{ {0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76},
 {0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xaa,0xaf,0x9c,0xa4,0x72,0xc0},
 {0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xaa,0x5e,0xf1,0x71,0xd8,0x31,0x15},
 {0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75},
 {0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84},
 {0x53,0x0c,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xeb,0xbe,0x39,0x4a,0x4c,0x58,0xef},
 {0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8},
 {0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2},
 {0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73},
 {0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb},
 {0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79},
 {0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08},
 {0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a},
 {0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e},
 {0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xcce,0x55,0x28,0xdf},
 {0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16}
}; //s-盒
```

```
byte InvSBox[16][16] =
{ {0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb},
 {0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb},
 {0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xec,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e},
 {0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25},
 {0x72,0xf8,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92},
 {0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84},
 {0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06},
 {0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0xf0,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b},
 {0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xc6,0x73},
 {0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e},
 {0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b},
 {0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0,0xfe,0x78,0xcd,0x5a,0xf4},
 {0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10,0x59,0x27,0x80,0xec,0x5f},
 {0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0xd0,0x2d,0xe5,0x7a,0x9f,0x93,0xc9,0x9c,0xef},
 {0xa0,0xe0,0x3b,0x4d,0xac,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61},
 {0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14,0x63,0x55,0x21,0x0c,0x7d},
 }; //逆s-盒
```

```
byte Rc[31] =
{ 0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
 0xfa, 0xef, 0xc5, 0x91};
```

AES 算法的主线是围绕一个名为 State 的数组实现的, 该数组是一个  $4 \times 4$  的二维数组, 一次可载入 16 个字节的数据, 所以 AES 算法的次加密/解密操作即为 16 个字节。通过主循环对 State 矩阵执行 4 个方法的操作, 在 AES 算法规范中称为 SubBytes (字节替换)、ShiftRows (行位移变换)、MixColumns (列混合变换) 和 AddRoundKey (轮密钥加)。AddRoundKey 使用从种子密钥值中生成的轮密钥代替 4 组字节。SubBytes 用一个代替表替换单个字节。ShiftRows 通过旋转 4 字节行的 4 组字节进行序列置换。MixColumns 用域加和域乘的组合来替换字节。

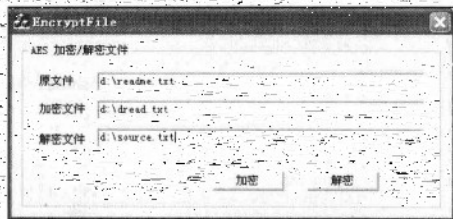


图 14.4 对档案进行加密和解密

## 实现过程

- (1) 新建一个基于对话框的工程。
- (2) 在窗体上添加 3 个文本编辑框控件和两个按钮控件。
- (3) 定义类 CAES 作为 AES 算法的实现类。类定义如下:

```
typedef enum ENUM_KeySize_ //密钥长度类型
{
 BIT128 = 0,
 BIT192,
 BIT256
}ENUM_KEYSIZE;

typedef enum GFCALCMODE_ //列混合变换模式
{
 MODE01 = 0,
 MODE02,
 MODE03,
 MODE09,
 MODE0b,
 MODE0d,
 MODE0e
}GFCALCMODE;

typedef struct BYTE4_
{
 BYTE w[4];
}BYTE4;

class CAES //AES加密算法类
{
private:
 int Nk,Nr; //Nk-密钥在调度表中所占行数; Nr-轮密钥加的运算次数-1
 byte (* State)[4],*w[4],*key[4]; //状态、密钥表、密钥

 void SubBytes(); //字节转换
 void ShiftRows(); //行位移变换
 void MixColumns(); //列混合运算
 void AddRoundKey(int round); //轮密钥加
 void KeyExpansion(); //生成调度表
 void InvShiftRows(); //反向行位移变换
 void InvSubBytes(); //反向字节转换
 void InvMixColumns(); //反向列混合运算
 BYTE GfCalc(BYTE b,GFCALCMODE Mode); //域运算方法
 void Encrypt(BYTE * input,BYTE * output); //加密16字节数据
 void Decrypt(BYTE * input,BYTE * output); //解密16字节数据
 void EncryptBuffer(BYTE * input,int length); //加密指定长度数据
 void DecryptBuffer(BYTE * input,int length); //解密指定长度数据

public:
 CAES();
 virtual ~CAES();

 bool SetKeys(ENUM_KEYSIZE KeySize,CString sKey); //设置密钥
 CString &EncryptString(CString &input); //加密字符串
 CString &DecryptString(CString &input); //解密字符串
 void EncryptFile(CString SourceFile,CString TagerFile); //加密文件
 void DecryptFile(CString SourceFile,CString TagerFile); //解密文件
};
```

- (4) 在 CAES 类中实现 SetKeys 方法,该方法用于指定密钥及轮密钥加运算的次数。实现代码如下:

```
bool CAES::SetKeys(ENUM_KEYSIZE KeySize,CString sKey)
{
 int i,j;
 switch(KeySize) //Nk为轮密钥加的次数减1
 {
 case BIT128:
 this->Nk = 4; //128÷8÷4
 this->Nr = 10; //Nr = Nk + 6
 break;
 case BIT192:
 this->Nk = 6; //192÷8÷4
 this->Nr = 12; //Nr = Nk + 6
 break;
 case BIT256:
 this->Nk = 8; //256÷8÷4
 this->Nr = 14; //Nr = Nk + 6
 break;
 default:
 this->Nk = 8;
 this->Nr = 14;
 break;
 }
```

```

 }
 for(i=0;i<4;i++)
 {
 if(key[i]!=NULL)
 {
 delete key[i];
 key[i]=NULL;
 }
 if(w[i]!=NULL)
 {
 delete w[i];
 w[i]=NULL;
 }
 }
 for(i=0;i<4;i++)
 {
 key[i]=new byte[Nk];
 if(key[i]==NULL)
 {
 return false;
 }
 }
 for(i=0;i<4;i++)
 {
 w[i]=new byte[4*(Nr+1)];
 if(w[i]==NULL)
 {
 return false;
 }
 }
 for(i=0;i<4;i++)
 {
 for(j=0;j<Nk;j++)
 key[i][j]=sKey.GetAt(Nk*i+j);
 }
 KeyExpansion();
 return true;
}

```

//释放原有的内存并置空

//动态创建密钥数组

//动态创建密钥调度表

//获得密钥

//生成密钥调度表

(5) 在 CAES 类中实现 KeyExpansion 方法，该方法用来生成密钥调度表。实现代码如下：

```

void CAES::KeyExpansion()
{
 byte temp[4],tp;
 int i,j,x;
 for(i=0;i<Nk;i++)
 {
 w[0][i]=key[0][i];
 w[1][i]=key[1][i];
 w[2][i]=key[2][i];
 w[3][i]=key[3][i];
 }
 while(i<4*(Nr+1))
 {
 for(j=0;j<4;j++)
 temp[j]=w[j][i-1];
 if(i%Nk==0)
 {
 tp=temp[0];
 temp[0]=temp[1];
 temp[1]=temp[2];
 temp[2]=temp[3];
 temp[3]=tp;
 for(j=0;j<4;j++)
 temp[j]=SBox[(temp[j]>>4)&0x0F][temp[j]&0x0F];
 x=Rc[i/Nk];
 temp[0]=temp[0]^x;
 }
 else
 {
 if(Nk>6&&(i%Nk==4))
 {
 for(j=0;j<4;j++)
 temp[j]=SBox[(temp[j]>>4)&0x0F][temp[j]&0x0F];
 }
 for(j=0;j<4;j++)
 w[j][i]=(w[j][i-Nk]^temp[j]);
 i++;
 }
 }
 //实现密钥的每16个字节为一个矩阵的矩阵转置(对字符串加密/解密有影响)
 for(int k=0;k<=Nr;k++)
 {
 for(i=0;i<3;i++)
 {
 x=4*k;
 for(j=i+1;j<4;j++)
 {
 tp=w[i][x+j];
 w[i][x+j]=w[j][x+i];
 w[j][x+i]=tp;
 }
 }
 }
}

```

//密钥扩展

//生成前Nk个字

//生成后几个字



(6) 在 CAES 类中实现 AddRoundKey 方法, 该方法用于轮密钥加法运算, 参数 round 决定了 State 矩阵与密钥调度表中的哪一行进行加法操作。实现代码如下:

```
void CAES::AddRoundKey(int round) //轮密钥加法变换
{
 int i,j,k=round*4;
 for(i=0;i<4;i++) //循环State矩阵与调度表中的数据相加
 for(j=0;j<4;j++)
 State[i][j]=State[i][j]^w[i][k+j]; //^代表加法运算
}
```

(7) 在 CAES 类中实现 SubBytes 方法, 该方法用于单字节代替变换操作。实现代码如下:

```
void CAES::SubBytes() //字节代替变换
{
 for(int i=0;i<4;i++)
 for(int j=0;j<4;j++)
 State[i][j]=SBox[(State[i][j]>>4)&0x0F][(State[i][j]&0x0F)];
}
```

(8) 在 CAES 类中实现 ShiftRows 方法, 该方法用于行位移变换操作。实现代码如下:

```
void CAES::ShiftRows() //循环移位
{
 BYTE4 temp[4];
 for (int r = 0; r < 4; ++r) //将State复制到临时矩阵
 {
 for (int c = 0; c < 4; ++c)
 {
 temp[r].w[c] = this->State[r][c];
 }
 }

 for (r = 1; r < 4; ++r) //位移变换操作
 {
 for (int c = 0; c < 4; ++c)
 {
 this->State[r][c] = temp[r].w[(c + r) % 4];
 }
 }
}
```

(9) 在 CAES 类中实现 MixColumns 方法, 该方法用于实现列混合变换操作, 也就是域乖加计算。实现代码如下:

```
void CAES::MixColumns() //列混合变换
{
 BYTE4 temp[4];
 for (int r = 0; r < 4; ++r) //将State复制到临时矩阵
 {
 for (int c = 0; c < 4; ++c)
 {
 temp[r].w[c] = this->State[r][c];
 }
 }

 for (int c = 0; c < 4; ++c) // 域乖加计算
 {
 this->State[0][c] = (BYTE) (
 (int)GfCalc(temp[0].w[c],MODE02) ^
 (int)GfCalc(temp[1].w[c],MODE03) ^
 (int)GfCalc(temp[2].w[c],MODE01) ^
 (int)GfCalc(temp[3].w[c],MODE01));
 this->State[1][c] = (BYTE) (
 (int)GfCalc(temp[0].w[c],MODE01) ^
 (int)GfCalc(temp[1].w[c],MODE02) ^
 (int)GfCalc(temp[2].w[c],MODE03) ^
 (int)GfCalc(temp[3].w[c],MODE01));
 this->State[2][c] = (BYTE) (
 (int)GfCalc(temp[0].w[c],MODE01) ^
 (int)GfCalc(temp[1].w[c],MODE01) ^
 (int)GfCalc(temp[2].w[c],MODE02) ^
 (int)GfCalc(temp[3].w[c],MODE03));
 this->State[3][c] = (BYTE) (
 (int)GfCalc(temp[0].w[c],MODE03) ^
 (int)GfCalc(temp[1].w[c],MODE01) ^
 (int)GfCalc(temp[2].w[c],MODE01) ^
 (int)GfCalc(temp[3].w[c],MODE02));
 }
}
```

(10) GfCalc 方法用来计算单字节的乖加计算, 并根据不同的计算模式进行不同的计算。实现



代码如下：

```
BYTE CAES::GfCalc(BYTE b,GFCALCMODE Mode)
{
 switch(Mode)
 {
 case MODE01:
 return b;
 break;
 case MODE02:
 if (b < 0x80)
 return (BYTE)(int)(b << 1);
 else
 return (BYTE)((int)(b << 1) ^ (int)(0x1b));
 break;
 case MODE03:
 return (BYTE)((int)GfCalc(b,MODE02) ^ (int)b); // (b*2)+b*1
 break;
 case MODE09:
 return (BYTE)((int)GfCalc(GfCalc(GfCalc(b,MODE02),MODE02),MODE02) ^ (int)b); // (b*2*2*2)+b*1
 break;
 case MODE0b:
 return (BYTE)((int)GfCalc(GfCalc(GfCalc(b,MODE02),MODE02),MODE02) ^ (int)GfCalc(b,MODE02) ^ (int)b); // (b*2*2*2)+(b*2)+b*1
 break;
 case MODE0d:
 return (BYTE)((int)GfCalc(GfCalc(GfCalc(b,MODE02),MODE02),MODE02) ^ (int)GfCalc(GfCalc(b,MODE02),MODE02) ^ (int)(b)); // (b*2*2*2)+(b*2*2)+b*1
 break;
 case MODE0e:
 return (BYTE)((int)GfCalc(GfCalc(GfCalc(b,MODE02),MODE02),MODE02) ^ (int)GfCalc(b,MODE02) ^ (int)GfCalc(b,MODE02)); // (b*2*2*2)+(b*2*2)+b*2
 break;
 default:
 return b;
 }
}
```

### 举一反三

根据本实例，读者可以：

- 对图片批量解密文件；
- 利用各种图片加密文件。

## 14.2 软件注册与加密

为了使开发的软件能被更广泛地使用，开发者希望更多地用户试用软件，而另一方面，开发者又不想让用户长时间免费使用未经授权的软件，这就需要设计软件注册程序。下面通过几个软件注册与加密的实例，介绍软件注册和加密的方法。

### 实例 391

#### 利用 INI 文件对软件进行注册

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\14\391

### 实例说明

在网络上销售的商业管理系统，一般都需注册方可使用。本实例将使用 INI 文件对软件进行用户信息注册。运行程序，如果在 INI 文件中有注册信息，在程序对话框的标题栏上将不显示“未注册”字样，如图 14.5 所示，相反如果没有注册信息就会该字样，如图 14.6 所示，单击“注册”按钮可以将注册码写入到 INI 文件中。

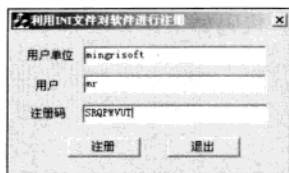


图 14.5 软件注册后程序运行界面

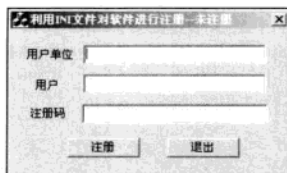


图 14.6 软件未注册时程序运行界面

## 技术要点

本实例主要通过 GetPrivateProfileString 函数来读取 INI 文件，读取 INI 文件中的注册码后，通过自定义函数 UnEncrypt 将注册码转换成机器码，然后和程序中预定义的机器码进行比较，如果相同，就去掉“未注册”字样。

未注册的程序可以通过“注册”按钮，调用 WritePrivateProfileString 函数将注册码写入到 INI 文件中。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在对话框上添加 3 个静态文本控件；添加 3 个文本编辑控件，设置 ID 属性分别为 IDC\_EDPART、IDC\_EDUSR 和 IDC\_EDREGCODE；添加两个按钮，设置 ID 属性分别为 IDC\_BTREG 和 IDC\_BTEXIT，Caption 属性分别为“注册”和“退出”。

(3) 在实现文件 INIRegDlg.cpp 中加入如下全局变量声明：

```
char machine[]="01234567"; //定义机器码
char susrpart[128];
char susrid[128];
char sregecode[128];
char path[128];
```

(4) 在 OnInitDialog 中通过读取 INI 文件判断程序是否已经注册，代码如下：

```
BOOL CINIRegDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 ::GetCurrentDirectory(128,path);
 strcat(path,"\\reg.ini");
 CString temp;
 //获得INI文件中数据
 GetPrivateProfileString("Registration","userpart","",susrpart,128,path);
 GetPrivateProfileString("Registration","userid","",susrid,128,path);
 GetPrivateProfileString("Registration","regcode","",sregecode,128,path);
 temp.Format("%s",machine);
 if(temp==UnEncrypt(sregecode))
 SetWindowText("利用INI文件对软件进行注册");
 return TRUE;
}
```

(5) 按钮“注册”的实现函数，将注册信息写入到 INI 文件中，并检验注册码是否正确，代码如下：

```
void CINIRegDlg::OnReg()
{
 //获得控件中数据
 CString struserpart,struserid,strregcode;
 GetDlgItem(IDC_EDPART)->GetWindowText(struserpart);
 GetDlgItem(IDC_EDUSR)->GetWindowText(struserid);
 GetDlgItem(IDC_EDREGCODE)->GetWindowText(strregcode);
 //写入到INI文件中
 WritePrivateProfileString(_T(struserpart),_T(struserid),_T(strregcode),
 _T(path));
 CString temp;temp=machine;
 if(temp==UnEncrypt(strregcode.GetBuffer(0)))
 {
 AfxMessageBox("注册成功");
 SetWindowText("利用INI文件对软件进行注册");
 }
}
```

(6) 自定义函数 UnEncrypt, 实现对字符串的加密和解密, 代码如下:

```
CString CINIRegDlg::UnEncrypt(char* strcode)
{
 CString temp;
 for(int i=0;i<8;i++)
 {
 strcode[i]=strcode[i]^1123;
 }
 temp=strcode;
 return temp;
}
```

### 举一反三

根据本实例, 读者可以:

- 在 INI 文件中保存加密过的注册信息;
- 开发使用 INI 文件保存密钥的应用程序。

## 实例 392

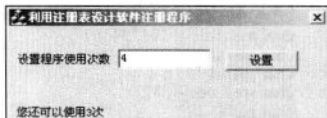
### 利用注册表设计软件注册程序

本实例可以提高计算机安全性

实例位置: 光盘\mingrisoft\14\392

### 实例说明

在注册表写入信息是注册软件常用的一种方法, 本实例通过在程序中设置使用次数来限制程序的使用。运行程序, 在窗体的底部显示程序还可以使用的次数, 如图 14.7 所示, 程序在一开始运行时将使用次数限制在 5 次, 但通过“设置”按钮



### 技术要点

本实例主要通过 RegCreateKey 函数创建或打开注册表项, 图 14.7 利用注册表设计软件注册程序通过 RegSetValueEx 函数设置注册表值项及值项数据。通过 RegQueryValueEx 函数获得值项的数据, 然后判断是否小于 1, 如果小于就停止程序的运行。

### 实现过程

(1) 新建一个名为 RegSoft 的对话框 MFC 工程。

(2) 在对话框上添加两个静态文本控件, 将其中一个控件的 ID 属性设置为 IDC\_SPARE, 并添加成员变量 m\_spare; 添加一个文本编辑控件, 设置 ID 属性为 IDC\_EDSET; 添加一个按钮控件, 设置 Caption 属性为“设置”。

(3) 在 OnInitDialog 函数中读取注册表中的信息, 判断程序是否能够运行, 代码如下:

```
BOOL CRegSoftDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 HKEY Key;
 CString sKeyPath;
 //使用次数的值项在注册表中所在位置
 sKeyPath="Software\\mingrisoft";
 if(RegOpenKey(HKEY_CURRENT_USER,sKeyPath,&Key)==2)
 {
 //在注册表中记录已试用的次数
 ::RegCreateKey(HKEY_CURRENT_USER,sKeyPath,&Key);
 ::RegSetValueEx(Key,"TryTime",0,REG_SZ,(unsigned char*)"5",2);
 ::RegCloseKey(Key);
 m_spare.SetWindowText("您还可以使用5次");
 }
 else //已经存在注册信息
 {
 CString sTryTime;
 int nTryTime;
```

```
LPBYTE Data=new BYTE[80];
DWORD TYPE=REG_SZ;
DWORD cbData=80;
//取出已记载的数量
::RegQueryValueEx(Key,"TryTime",0,&TYPE,Data,&cbData);
sTryTime.Format("%s",Data);
nTryTime=atoi(sTryTime);
if(nTryTime<1)
{
 MessageBox("您的最大试用次数已过，只有注册后才允许继续使用！",
 "系统提示",MB_OK|MB_ICONSTOP);
 return FALSE;
}
nTryTime--;
sTryTime.Format("%d",nTryTime);
::RegSetValueEx(Key,"TryTime",0,REG_SZ,(unsigned char*)sTryTime.GetBuffer(0),2);
::RegCloseKey(Key);
CString temp;
temp.Format("您还可以使用%d次",nTryTime);
m_spare.SetWindowText(temp);
}
return TRUE;
```

(4) 按钮“设置”的实现函数，该函数可以修改程序的试用次数，代码如下：

```
void CRegSoftDlg::OnSet()
{
 HKEY Key;
 CString str;
 CString sKeyPath;
 sKeyPath="Software\\mingrisoft";
 GetDlgItem(IDC_EDSET)->GetWindowText(str); //获得控件中数据
 ::RegCreateKey(HKEY_CURRENT_USER,sKeyPath,&Key); //打开注册表键
 ::RegSetValueEx(Key,"TryTime",0,REG_SZ,(unsigned char*)str.GetBuffer(0),str.GetLength()); //设置键值
 ::RegCloseKey(Key); //关闭注册表
 MessageBox("设置成功","系统提示",MB_OK|MB_ICONSTOP);
 GetDlgItem(IDC_EDSET)->SetWindowText("");
}
```

### 举一反三

根据本实例，读者可以：

- 将注册信息加密后存入注册表；
- 开发将控件的注册信息写到注册表中的程序。

## 实例 393

### 利用网卡序列号设计软件注册程序

本实例可以提高计算机安全性

实例位置：光盘\mingrisoft\14\393

### 实例说明

在市场或网络上销售的商业管理软件，用户在使用时都需要进行注册，软件被注册后用户才有权使用。本实例将利用网卡的序列号生成注册码，运行程序，在对话框上面的编辑框中显示出网卡的序列号，通过“生成序列号”按钮来生成注册码，显示在下面的文本框中，如图 14.8 所示。

### 技术要点

本实例主要通过 Netbios 函数来获得网卡的序列号。网卡序列号一般为 3 段，在程序中分别将这 3 段存储在 3 个 char 数组中，然后该数组中的字符转换成它们所对应的整型数值，再将根据转换得出的数值通过加密算法得出另外的数值并转换成 ASCII 码字符，最后将所有的 ASCII 码字符合并就形成了注册码。

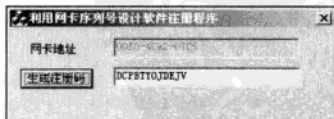


图 14.8 利用网卡序列号设计软件注册程序

### 实现过程

(1) 新建一个名为 NetMACReg 的对话框 MFC 工程。



(2) 在对话框上添加两个文本编辑控件, 设置 ID 属性分别为 IDC\_MACADDR 和 IDC\_EDREGCODE; 添加一个按钮控件, 设置 ID 属性为 IDC\_BTREG, Caption 属性为“生成注册码”。

(3) 在 OnInitDialog 函数中获得网卡的序列号, 代码如下:

```
BOOL CNetMACRegDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 NCB ncb;
 LAN_ENUM lenum;
 ADAPTER_STATUS state;
 UCHAR ucReturnCode;
 ncb.ncb_command=NCBENUM;
 ncb.ncb_buffer=(UCHAR*)&lenum;
 ncb.ncb_length=sizeof(lenum);
 ucReturnCode=Netbios(&ncb);
 if(lenum.length>=0)
 {
 int num=lenum.lana[0];
 UCHAR buf[128];
 memset(&ncb,0,sizeof(ncb));
 ncb.ncb_command=NCBRESET;
 ncb.ncb_lana_num=num;
 ucReturnCode=Netbios(&ncb);
 memset(&ncb,0,sizeof(ncb));
 ncb.ncb_command=NCBASTAT;
 ncb.ncb_lana_num=num;
 ncb.ncb_buffer=(unsigned char *)&state;
 ncb.ncb_length=sizeof(state);
 strcpy((char *)ncb.ncb_callname,"*");
 ucReturnCode=Netbios(&ncb);
 CString strMac;
 //格式化序列号
 strMac.Format("%02X%02X-%02X%02X-%02X%02X\n",
 state.adapter_address[0],
 state.adapter_address[1],
 state.adapter_address[2],
 state.adapter_address[3],
 state.adapter_address[4],
 state.adapter_address[5]);
 GetDlgItem(IDC_MACADDR)->EnableWindow(false); //设置控件不可用
 GetDlgItem(IDC_MACADDR)->SetWindowText(strMac); //设置显示文本
 }
 return TRUE;
}
```

(4) 按钮“生成注册码”的实现函数, 该函数用于将网卡序列号生成注册码, 代码如下:

```
void CNetMACRegDlg::OnReg()
{
 CString code;
 CString regcode,tmp;
 GetDlgItem(IDC_MACADDR)->GetWindowText(code);
 code.MakeLower();
 CString seg1,seg2,seg3;
 int num;
 seg1=code.Mid(0,4);
 seg2=code.Mid(5,4);
 seg3=code.Mid(10,4);
 char *cpseg1=new char[4];
 char *cpseg2=new char[4];
 char *cpseg3=new char[4];
 cpseg1=seg1.GetBuffer(0);
 cpseg2=seg2.GetBuffer(0);
 cpseg3=seg3.GetBuffer(0);
 char temp;
 int i;
 //加密算法是将char数组中的字符转换成十进制数, 再乘以4加该字符在数组中的索引值
 for(i=0;i<4;i++)
 {
 temp=cpseg1[i];
 if(temp>='a'&&temp<='f')
 num=temp-'a'+10;
 else
 num=temp-'0';
 tmp.Format("%c",base[num*4+i]);
 regcode+=tmp;
 }
}
```

```
for(i=0;i<4;i++)
{
 temp=cpseg2[i];
 if(temp>='a'&&temp<='f')
 num=temp-'a'+10;
 else
 num=temp-'0';
 tmp.Format("%c",base[num*4+i]);
 regcode+=tmp;
}
for(i=0;i<4;i++)
{
 temp=cpseg3[i];
 if(temp>='a'&&temp<='f')
 num=temp-'a'+10;
 else
 num=temp-'0';
 tmp.Format("%c",base[num*4+i]);
 regcode+=tmp;
}
regcode.MakeUpper();
GetDlgItem(IDC_EDREGCODE)->SetWindowText(regcode);
}
```

### 举一反三

根据本实例，读者可以：

- 开发远程软件产品注册程序。

### 实例 394

### 根据 CPU 和磁盘序列号 设计软件注册程序

本实例可以提高计算机安全性

实例位置：光盘\mingrisoft\14\394

### 实例说明

软件注册过程中一个关键问题是如何生成每个用户都各不相同的机器码，本实例通过 CPU 和磁盘序列号生成一个机器码，然后根据该机器码生成注册号。运行程序，在对话框中显示 CPU、磁盘序列号和生成后的机器码，单击“生成注册号”按钮计算得出最终的注册号，如图 14.9 所示。

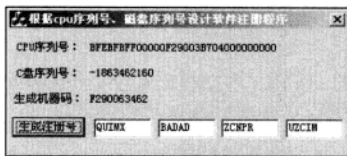


图 14.9 根据 CPU 和磁盘序列号设计软件注册程序

### 技术要点

本实例生成机器码的算法是提取 CPU 序列号的第 13 位~第 18 位和 C 盘序列号的第 3 位~第 8 位，顺序连接形成一个 10 个字符的字符串，该字符串就是机器码。通过机器码计算注册号主要是计算出机器码字符串中每个字符所对应的十进制数，以该十进制数作为索引在数组中选择字符，然后合并所有的字符形成注册码，最后将注册码字符串分成 4 组显示在编辑框中。

### 实现过程

- (1) 新建一个名为 CPUAndDiskReg 的对话框 MFC 工程。
- (2) 在对话框上添加 6 个静态文本控件，将其中 3 个 ID 属性设为 IDC\_CPU、IDC\_CDISK 和 IDC\_MACHINE，并添加成员变量 m\_cpu、m\_cdisk 和 m\_machine；添加 4 个文本编辑控件；添加 1 个按钮控件，设置 ID 属性为 IDC\_REG，Caption 属性为“生成注册号”。
- (3) 在 OnInitDialog 函数中获得 CPU 和磁盘序列号，然后生成机器码，代码如下：

```
BOOL CCPUAndDiskRegDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
 ...//此处代码省略
 //获取CPU序列号
```

```

unsigned long s1,s2;
char sel;
sel='1';
CString MyCpuID,CPUID1,CPUID2;
__asm{
mov eax,01h
xor edx,edx
cpuid
mov s1,edx
mov s2,eax
}
CPUID1.Format("%08X%08X",s1,s2);
__asm{
mov eax,03h
xor ecx,ecx
xor edx,edx
cpuid
mov s1,edx
mov s2,ecx
}
CPUID2.Format("%08X%08X",s1,s2);
MyCpuID=CPUID1+CPUID2;
m_cpu.SetWindowText(MyCpuID);
DWORD ser;
char namebuf[128];
char filebuf[128];
//获取C盘的序列号
::GetVolumeInformation("c:\\",namebuf,128,&ser,0,0,filebuf,128);
CString strdisk;
strdisk.Format("%d",ser);
CString strmachine;
strmachine=MyCpuID.Mid(13,5); //从MyCpuID的第13位开始取5个
strmachine+=strdisk.Mid(3,5); //从strdisk的第3位开始取5个,合并生成机器码
m_cdisk.SetWindowText(strdisk);
m_machine.SetWindowText(strmachine);
return TRUE;
}

```

(4) 按钮“生成注册号”的实现函数, 根据机器码生成用来注册的注册码, 代码如下:

```

void CCPUAndDiskRegDlg::OnReg()
{
 //定义一个密钥数组
 CString code[16]={ "ad","eh","im","np","ru","vy","zc","gk",
 "pt","xb","fj","ox","wa","ei","nr","qu"};
 CString reg,stred;
 int num;
 m_machine.GetWindowText(stred);
 stred.MakeLower();
 //根据十六进制数字从密钥数组中选择字符串
 for(int i=0;i<10;i++)
 {
 char p=stred.GetAt(i);
 if(p>='a'&&p<='f')
 num=p-'a'+10;
 else
 num=p-'0';
 CString tmp=code[num];
 reg+=tmp;
 }
 reg.MakeUpper();
 GetDlgItem(IDC_NUM1)->SetWindowText(reg.Mid(0,5));
 GetDlgItem(IDC_NUM2)->SetWindowText(reg.Mid(5,5));
 GetDlgItem(IDC_NUM3)->SetWindowText(reg.Mid(10,5));
 GetDlgItem(IDC_NUM4)->SetWindowText(reg.Mid(15,5));
}

```

## 举一反三

根据本实例, 读者可以:

- 给销售的软件产品授权。

## 第 15 章

### 实用工具

- 实现纪念日提醒
- SQL 数据库提取器
- 加班网上管理
- 垃圾文件清理工具
- 网页照相机
- 屏幕截图工具

Visual C++



本节中笔者设计了一些小工具，包括桌面日历、加班网上管理，垃圾文件清理工具等。这些小工具会对您的日常学习或生活有所帮助。

### 实例 395 实现纪念日提醒

本实例可以提高基础技能

实例位置：光盘\mingrisoft\15\395

#### 实例说明

每个用户在一生之中都会有一些具有特殊意义的日期，这些日期称之为纪念日，而随着年龄的增长，纪念日的数量也会逐渐的增加，想要全部记得，确实是很费神的事，本实例就是一款可以对设计好的纪念日进行定时提醒的工具，可以帮助用户更好的管理纪念日，实例运行效果如图 15.1 所示。



图 15.1 纪念日提醒

#### 技术要点

本实例通过定时器来实现日期的定时判断，下面就介绍定时器的用法。在设置定时器时，可以使用 SetTimer 方法。

SetTimer 方法用于设置定时器，其语法格式如下：

```
UINT SetTimer(UINT nIDEvent, UINT nElapsed, void (CALLBACK EXPORT* lpfnTimer)(HWND, UINT, UINT, DWORD));
```

参数说明：

- nIDEvent：定时器标识索引。
- nElapsed：定时器的间隔时间，以毫秒为单位。
- lpfnTimer：指定了应用程序提供的 TimerProc 回调函数的地址，该函数被用于处理 WM\_TIMER 消息。如果这个参数为 NULL，则 WM\_TIMER 消息被放入应用程序的消息队列并由 CWnd 对象来处理。

#### 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 单击“Insert”/“Resource”菜单项，在打开的 Insert Resource 对话框中单击 Import 按钮，向工程中导入位图资源。

(3) 向对话框中添加一个图片控件、一个月历控件、一个列表视图控件和一个按钮控件。右键单击图片控件,在弹出的菜单中选择 Properties 选项,设置 Type 属性为 Bitmap,设置 Image 属性为 IDB\_BITMAP1。设置列表视图控件的显示风格为 Report 风格。

(4) 处理对话框的定时器事件,在定时器中判断当前日期是否为用户设置的纪念日日期,如果是则弹出消息对话框进行提醒,代码如下:

```
void CCommemorateDlg::OnTimer(UINT nIDEvent)
{
 UpdateData();
 CTime time = CTime::GetCurrentTime();
 for (int i=0;i<m_List.GetItemCount();i++)
 {
 CString strTime = m_List.GetItemText(i,1);
 if (time.Format("%m-%d") == strTime)
 {
 CString strName = m_List.GetItemText(i,0);
 MessageBox("今天是: "+strName,"提示");
 }
 }
 CDialog::OnTimer(nIDEvent);
}
```

//获得当前系统日期  
//根据列表项数量进行循环  
//获得纪念日对应日期  
//当前日期是否和纪念日相同  
//获得纪念日名称  
//纪念日提示

### 举一反三

根据本实例,读者可以:

- 创建闹钟程序。

## 实例 396 SQL 数据库提取器

本实例是一个提高基础技能的程序

实例位置: 光盘\mingrisoft\15\396

### 实例说明

SQL 数据库提取器可以对 SQL 数据库进行简单的操作,使用户能够更方便地操作 SQL 数据库。SQL 数据库提取器可以连接获得服务器中所有的数据库,用户可以选择数据库中的表,并提取所选择表的结构信息,并可以分别向 WORD 文档和 EXCEL 表格中插入数据,本实例通过 Visual C++ 实现了 SQL 数据库提取器的功能。运行本实例,选择服务器,并设置用户名和密码,单击“登录”按钮,即可登录当前选择的服务器,并获得服务器中所有数据库的信息,选择一个数据库及数据库中的数据表,该表的结构信息将显示在下方的列表中,单击“提取到 WORD”按钮,可以将该表的结构信息导入到 WORD 文档中,单击“提取到 EXCEL”按钮,可以将该表的结构信息导入到 EXCEL 表格中,程序运行结果如图 15.2 所示。

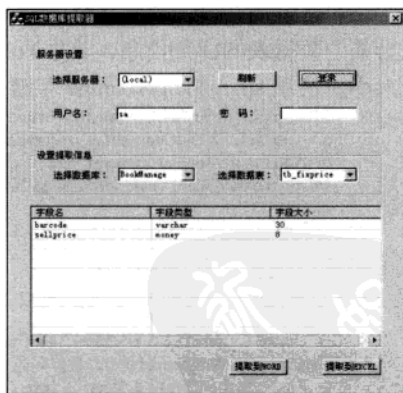


图 15.2 SQL 数据库提取器

### 技术要点

本示例中实现 SQL 数据库提取器时,主要用 SQL 语句和 sp\_mshelpcolumns 存储过程,实现了获得服务器中的数据库、数据表和表结构,下面对本实例中用到的关键技术进行详细讲解。

(1) 获得服务器中的数据库。在 SQL Server 数据库中, 每个服务器都包含一个名为 Master 的系统数据库, 在该数据库中的 sysdatabases 数据表中, 存储了当前服务器中所有的数据库名称, 本实例中用于获得服务器中数据库的代码如下:

```
_bstr_t bstrSQL;
bstrSQL = "select*from sysdatabases where dbid>6"; //设置查询语句
CString strText;
m_Ado.m_pRecordset.CreateInstance(__uuidof(Recordset)); //记录集对象实例化
m_Ado.m_pRecordset->Open(bstrSQL,m_Ado.m_pConnection.GetInterfacePtr(),
adOpenDynamic,adLockOptimistic,adCmdText); //打开记录集
while(!m_Ado.m_pRecordset->adoEOF) //循环读取记录集
{
 //获取数据库名称
 strText=(char*)(_bstr_t)m_Ado.m_pRecordset->GetCollect("name");
 m_Database.AddString(strText); //插入到控件中
 m_Ado.m_pRecordset->MoveNext(); //向下移动记录集指针
}
m_Ado.ExitConnect(); //断开数据库连接
```

(2) 获得数据库中的数据表。在每个数据库中都包含一个 sysobjects 数据表, 该表保存了数据库中的数据表、视图、存储过程等信息, 用户可以通过查询语句获得当前数据库中的数据表信息, 本实例中用于获得数据库中的数据表的代码如下:

```
m_Ado.OnInitADOConn(database,server,name,password); //连接数据库
CString bstrSQL;
bstrSQL.Format("select*from sysobjects where xtype='U'"); //设置查询语句
CString strText;
m_Ado.m_pRecordset.CreateInstance(__uuidof(Recordset)); //实例化记录集对象
m_Ado.m_pRecordset->Open((_bstr_t)bstrSQL,m_Ado.m_pConnection.GetInterfacePtr(),
adOpenDynamic,adLockOptimistic,adCmdText); //打开记录集
int i=0;
while(m_Ado.m_pRecordset->adoEOF==0) //循环读取记录集数据
{
 strText=(char*)(_bstr_t)m_Ado.m_pRecordset->GetCollect("name"); //获得数据表名
 m_Table.InsertString(i++,strText); //插入到控件中
 m_Ado.m_pRecordset->MoveNext(); //向下移动记录集指针
}
m_Ado.ExitConnect(); //断开数据库连接
```

(3) sp\_mshelpcolumns 存储过程。在 SQL Server 中提供了 sp\_mshelpcolumns 存储过程用于查看某个数据表的详细信息, 其中 col\_name 是字段名、col\_type 是字段类型、col\_len 是字段大小。使用 sp\_mshelpcolumns 存储过程获得数据库中数据表的结构信息的代码如下:

```
sql.Format("sp_mshelpcolumns %s",table); //设置SQL语句
m_Ado.OnInitADOConn(database,server,name,password); //连接数据库
m_Ado.m_pRecordset->Open((_bstr_t)sql,m_Ado.m_pConnection.GetInterfacePtr(),
adOpenDynamic,adLockOptimistic,adCmdText); //打开记录集
int i=0;
while(!m_Ado.m_pRecordset->adoEOF) //循环读取记录
{
 m_Frame.InsertItem(i, ""); //向控件中插入行
 m_Frame.SetItemText(i,0,(char*)(_bstr_t)m_Ado.m_pRecordset->
 GetCollect("col_name")); //获得字段名
 m_Frame.SetItemText(i,1,(char*)(_bstr_t)m_Ado.m_pRecordset->
 GetCollect("col_type")); //获得字段类型
 m_Frame.SetItemText(i,2,(char*)(_bstr_t)m_Ado.m_pRecordset->
 GetCollect("col_len")); //获得自动大小
 i++;
 m_Ado.m_pRecordset->MoveNext(); //向下移动记录集指针
}
m_Ado.ExitConnect(); //断开数据库连接
```

## 实现过程

(1) 新建一个基于对话框的应用程序, 将其窗体标题改为“SQL 数据库提取器”。

(2) 向对话框中添加 2 个群组控件、5 个静态文本控件、2 个编辑框控件、3 个组合框控件、1 个列表视图控件和 4 个按钮控件。对话框中主要用到的控件及说明如表 15.1 所示。

表 15.1

对话框主要用到的控件及说明

控件 ID	属性设置	关联变量
IDC_EDIT1	无	CEdit m_Name
IDC_EDIT2	无	CEdit m_Pwd
IDC_COMBO1	无	CcomboBox m_Server
IDC_COMBO2	无	CcomboBox m_Database
IDC_COMBO3	无	CcomboBox m_Table
IDC_LIST1	View 属性: Report、勾选 Single Selection、Client edge 属性	CListCtrl m_Frame
IDC_REFURBISH	Caption: 刷新	无
IDC_LOGIN	Caption: 登录	无
IDC_TOWORD	Caption: 提取到 WORD	无
IDC_TOEXCEL	Caption: 提取到 EXCEL	无

(3) 在对话框中新建一个名为 CADO 的类, 该类用于连接数据库。

(4) 通过类向导向工程导入操作 WORD 文档和 EXCEL 表格的类。操作 WORD 文档需要导入的类包括 Application、Documents、\_Document、Range、Tables 和 Selection。操作 EXCEL 表格需要导入的类包括 Application、Workbooks、\_Workbook、Worksheets、\_Worksheet 和 Range。

(5) 在 StdAfx.h 头文件中导入 ADO 动态链接库, 其实现代码如下:

```
#import "C:\Program Files\Common Files\System\ado\msado15.dll" no_namespace
rename("EOF","adoEOF")rename("BOF","adoBOF")
```

(6) 在程序类的 InitInstance 函数中初始化 COM 环境, 并获得当前程序所在路径, 代码如下:

```
GetCurrentDirectory(MAX_PATH,path); //获得程序所在路径
::CoInitialize(NULL); //初始化COM环境
```

(7) 在 CADO 类的头文件中声明 ADO 对象的智能指针, 其实现代码如下:

```
_ConnectionPtr m_pConnection; //连接对象智能指针
_RecordsetPtr m_pRecordset; //记录集对象智能指针
```

(8) 在 CADO 类中添加 OnInitADOConn 函数, 用于连接数据库, 其实现代码如下:

```
BOOL CADO::OnInitADOConn(CString database, CString server, CString name, CString password)
{
 CString strname; //声明字符串
 try
 {
 if(name.IsEmpty()) //用户名不能为空
 {
 AfxMessageBox("请输入用户!");
 return FALSE;
 }
 else
 {
 strname.Format("Provider=SQLOLEDB.1;Persist Security
 Info=False;User ID=%s;\
 pwd=%s;Initial Catalog=%s;Data Source=%s",name,password,database,server); //设置连接语句
 m_pConnection.CreateInstance("ADODB.Connection"); //初始化连接对象实例
 _bstr_t strConnect=strname;
 m_pConnection->Open(strConnect,"","",adModeUnknown); //连接数据库
 }
 return TRUE;
 }
 catch(_com_error e) //捕捉可能出现的错误
 {
 AfxMessageBox(e.Description());
 return FALSE;
 }
}
```

(9) 在 CADO 类中添加 ExitConnect 函数, 该函数用于断开数据库连接, 其实现代码如下:

```
void CADO::ExitConnect()
{
 if(m_pRecordset!=NULL) //如果记录集不为空
 m_pRecordset->Close(); //关闭记录集
 m_pConnection->Close(); //断开数据库连接
}
```



(10) 在主窗口中添加自定义函数 GetServer, 该函数用于获得数据库中的服务器信息, 其实现代码如下:

```
CString CDistillSQLDlg::GetServer()
{
 CString sSQLChar = "Driver={SQL Server}"; //声明字符串
 CString cKey = "SERVER"; //声明字符串
 SQLHENV hSqlHenv; // SQLHENV结构变量
 SQLHDBC hSQLHdbc; // SQLHDBC结构变量
 short sConnStrOut;
 CString Returnstr;
 //分配环境句柄
 int IsSuccess=SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&hSqlHenv);
 if (IsSuccess == SQL_SUCCESS || IsSuccess == SQL_SUCCESS_WITH_INFO)
 {
 //设置环境属性
 IsSuccess = SQLSetEnvAttr(hSqlHenv, SQL_ATTR_ODBC_VERSION,
 (void *)SQL_OV_ODBC3, 0);
 if (IsSuccess == SQL_SUCCESS || IsSuccess == SQL_SUCCESS_WITH_INFO)
 {
 //分配一个连接句柄
 IsSuccess = SQLAllocHandle(SQL_HANDLE_DBC, hSqlHenv, &hSQLHdbc);
 if (IsSuccess == SQL_SUCCESS || IsSuccess == SQL_SUCCESS_WITH_INFO)
 {
 CString szConnStrOut;
 //调用SQLBrowseConnect
 IsSuccess = SQLBrowseConnect(hSQLHdbc, (SQLCHAR *)sSQLChar.GetBuffer(sSQLChar.GetLength()),
 ,SQL_NTS,(SQLCHAR*)(szConnStrOut.GetBuffer(4824)), 4824, &sConnStrOut);
 szConnStrOut.ReleaseBuffer(); //释放控件
 int nPos=szConnStrOut.Find(cKey); //查找字符串SERVER
 if(nPos!=-1)
 {
 nPos=nPos+cKey.GetLength();
 int nBegin=szConnStrOut.Find("{",nPos+1); //查找字符串{
 int nEnd=szConnStrOut.Find(")",nPos+1); //查找字符串)
 Returnstr=szConnStrOut.Mid(nBegin+1,nEnd-(nBegin+1)); //截取字符串
 }
 }
 }
 return Returnstr; //返回服务器字符串
 }
}
```

(11) 在主窗口中添加自定义函数 FormatString, 该函数用于格式化服务器字符串, 其实现代码如下:

```
void CDistillSQLDlg::FormatString(CString sText, CComboBox *pComboBox)
{
 int nPos=0,nOldPos=0;
 CString sMem;
 while(nPos!=-1)
 {
 nOldPos=nPos;
 nPos=sText.Find(",",nPos+1); //查找“,”
 if(nPos==-1)
 {
 nPos=sText.GetLength(); //获得字符串长度
 if(nOldPos==0) //获得服务器名
 {
 sMem=sText.Mid(nOldPos,nPos-nOldPos);
 }
 else
 {
 sMem=sText.Mid(nOldPos+1,nPos-nOldPos-1);
 }
 if(nPos==sText.GetLength()) //如果拆分完服务器名
 {
 nPos=-1; //退出循环
 }
 if(sMem.IsEmpty())
 {
 continue;
 }
 pComboBox->AddString(sMem);
 }
 }
}
```

(12) 在主窗口中初始化时, 设置列表视图控件的扩展风格, 并设置表头信息, 调用 GetServer 函数和 FormatString 函数获得服务器字符串, 其实现代码如下:

```
CString sServer; //声明字符串
CString sText=GetServer(); //获得服务器名称
FormatString(sText,&m_Server); //格式化服务器名称
m_Server.SetCurSel(0); //设置服务器组合框显示选项
```

```

m_Name.SetWindowText("sa"); //设置默认用户名
//设置列表视图的扩展风格
m_Frame.SetExtendedStyle(LVS_EX_FLATSB //扁平风格显示滚动条
|LVS_EX_FULLROWSELECT //允许整行选中
|LVS_EX_HEADERDRAGDROP //允许整列拖动
|LVS_EX_ONECLICKACTIVATE //单击选中项
|LVS_EX_GRIDLINES); //画出网格线
//设置表头
m_Frame.InsertColumn(0,"字段名",LVCFMT_LEFT,155,0);
m_Frame.InsertColumn(1,"字段类型",LVCFMT_LEFT,155,1);
m_Frame.InsertColumn(2,"字段大小",LVCFMT_LEFT,155,2);

```

(13) 在主窗口中添加自定义函数 GetTableFrame，该函数用于获得数据表的结构信息，其实现代码如下：

```

void CDistillSQLDlg::GetTableFrame()
{
 m_Frame.DeleteAllItems(); //清空列表中数据
 CString sql,database,server,name,password,table;
 m_Table.GetLBText(m_Table.GetCurSel(),table); //获得表名
 sql.Format("sp_mshelpcolumns %s",table); //设置SQL语句
 m_Server.GetLBText(m_Server.GetCurSel(),server); //获得服务器名
 m_Database.GetLBText(m_Database.GetCurSel(),database); //获得数据库名
 m_Name.GetWindowText(name); //获得用户名
 m_Pwd.GetWindowText(password); //获得密码
 m_Ado.OnInitADOConn(database,server,name,password); //连接数据库
 m_Ado.m_pRecordset->Open((_bstr_t)sql,m_Ado.m_pConnection.GetInterfacePtr(), //获得记录集
 adOpenDynamic,adLockOptimistic,adCmdText);
 int i=0;
 while(!m_Ado.m_pRecordset->adoEOF) //循环读取记录
 {
 m_Frame.InsertItem(i,""); //向控件中插入行
 m_Frame.SetItemText(i,0,(char*)(_bstr_t)m_Ado.m_pRecordset->
 GetCollect("col_name")); //获得字段名
 m_Frame.SetItemText(i,1,(char*)(_bstr_t)m_Ado.m_pRecordset->
 GetCollect("col_type")); //获得字段类型
 m_Frame.SetItemText(i,2,(char*)(_bstr_t)m_Ado.m_pRecordset->
 GetCollect("col_len")); //获得自动大小
 i++;
 m_Ado.m_pRecordset->MoveNext(); //向下移动记录集指针
 }
 m_Ado.ExitConnect(); //断开数据库连接
}

```

(14) 在主窗口中添加自定义函数 GetTableList，该函数用于获得所选数据库中的数据表名称，其实现代码如下：

```

void CDistillSQLDlg::GetTableList()
{
 CString database,server,name,password;
 m_Database.GetLBText(m_Database.GetCurSel(),database); //获得数据库名称
 m_Server.GetLBText(m_Server.GetCurSel(),server); //获得服务器名称
 m_Name.GetWindowText(name); //获得用户名
 m_Pwd.GetWindowText(password); //获得密码
 m_Ado.OnInitADOConn(database,server,name,password); //连接数据库
 CString bstrSQL;
 bstrSQL.Format("select*from sysobjects where xtype='U'"); //设置查询语句
 CString strText;
 m_Ado.m_pRecordset.CreateInstance(__uuidof(Recordset)); //实例化记录集对象
 m_Ado.m_pRecordset->Open((_bstr_t)bstrSQL,m_Ado.m_pConnection.GetInterfacePtr(),
 adOpenDynamic,adLockOptimistic,adCmdText); //打开记录集
 int i=0;
 while(m_Ado.m_pRecordset->adoEOF==0) //循环读取记录集数据
 {
 strText=(char*)(_bstr_t)m_Ado.m_pRecordset->GetCollect("name"); //获得数据表名
 m_Table.InsertString(i++,strText); //插入到控件中
 m_Ado.m_pRecordset->MoveNext(); //向下移动记录集指针
 }
 m_Ado.ExitConnect(); //断开数据库连接
 m_Table.SetCurSel(0); //设置默认选择的表
 GetTableFrame(); //获得表结构
}

```



(15) 处理“登录”按钮的单击事件，在该事件的处理函数中连接数据库，其实现代码如下：

```
void CDistillSQLDlg::OnLogin()
{
 m_Table.ResetContent(); //清空数据表组合框
 m_Database.ResetContent(); //清空数据库组合框
 CString strname,server,name,password;
 m_Server.GetWindowText(server); //获得服务器名称
 m_Name.GetWindowText(name); //获得用户名
 m_Pwd.GetWindowText(password); //获得密码
 BOOL isConnection;
 if(server.IsEmpty()) //判断是否选择了服务器
 {
 MessageBox("请选择服务器!");
 return;
 }
 else
 {
 //连接数据库
 isConnection = m_Ado.OnInitADOConn("Master",server,name,password);
 }
 if(isConnection) //如果已连接数据库
 {
 _bstr_t bstrSQL;
 bstrSQL = "select* from sysdatabases where dbid>6"; //设置查询语句
 CString strText;
 m_Ado.m_pRecordset.CreateInstance(__uuidof(Recordset)); //记录集对象实例化
 m_Ado.m_pRecordset->Open(bstrSQL,m_Ado.m_pConnection.GetInterfacePtr(),
 adOpenDynamic,adLockOptimistic,adCmdText); //打开记录集
 while(!m_Ado.m_pRecordset->adoEOF) //循环读取记录集
 {
 //获取数据库名称
 strText=(char*)(_bstr_t)m_Ado.m_pRecordset->GetCollect("name");
 m_Database.AddString(strText); //插入到控件中
 m_Ado.m_pRecordset->MoveNext(); //向下移动记录集指针
 }
 m_Ado.ExitConnect(); //断开数据库连接
 m_Database.SetCurSel(0); //设置默认显示数据库
 GetTableList(); //获得数据库中的表
 }
}
```

(16) 处理“提取到 WORD”按钮的单击事件，在该事件的处理函数中将列表中的数据表结构信息导入到 WORD 文档中，其实现代码如下：

```
void CDistillSQLDlg::OnToword()
{
 _Application1 app; //WORD启动对象
 Documents doc; //文档管理对象
 CComVariant a (_T(""),b(false),c(0),d(true),aa(1),bb(20);
 _Document doc1; //文档对象
 Tables tabs; //表格
 Range1 rangestar,range; //选区
 Selection sele; //光标
 COleVariant colevariant;
 //初始化连接
 app.CreateDispatch("word.Application");
 doc.AttachDispatch(app.GetDocuments());
 doc1.AttachDispatch(doc.Add(&a,&b,&c,&d));
 range.AttachDispatch(doc1.GetContent());
 tabs.AttachDispatch(doc1.GetTables());
 tabs.Add(range,m_Frame.GetItemCount()+1,3,colevariant,colevariant); //创建表格
 sele.AttachDispatch(app.GetSelection()); //获得光标位置
 CString sText[]={"字段名","字段类型","字段大小"}; //设置标题字符串
 for(long num=0;num<3;num++) //循环表格列
 {
 sele.TypeText(sText[num]); //插入数据
 }
 //移动光标
}
```

```

sele.MoveRight((COleVariant)"1",(COleVariant)"1",(COleVariant)"0");
}
for(int i=0;i<m_Frame.GetItemCount();i++)
{
for(long j=0;j<3;j++)
{
sele.TypeText(m_Frame.GetItemText(i,j)); //插入表文
sele.MoveRight((COleVariant)"1",(COleVariant)"1",(COleVariant)"0"); //移动光标
}
}
app.SetVisible(true); //显示WORD文档
//是否使用的WORD对象
tabs.ReleaseDispatch();
sele.ReleaseDispatch();
doc.ReleaseDispatch();
doc1.ReleaseDispatch();
app.ReleaseDispatch();
}

```

(17) 处理“提取到 EXCEL”按钮的单击事件，在该事件的处理函数中将列表中的数据表结构信息导入到 EXCEL 表格中，其实现代码如下：

```

void CDistillSQLDlg::OnToexcel()
{
 CString strPath,strText="",strFile,name,table;
 m_Table.GetLBText(m_Table.GetCurSel(),table); //获得表名
 CFileDialog file(FALSE,NULL,table,OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,
 "EXCEL 文件(*.xls)*.xls|",AfxGetMainWnd()); //设置文件保存路径
 if(file.DoModal()==IDOK) //显示另存为对话框
 {
 strPath=file.GetPathName(); //获得文件路径
 if(strPath.Right(4) != ".xls") //如果没有扩展名
 strPath += ".xls"; //设置扩展名
 _Application2 app; //EXCEL启动对象
 Workbooks books; //工作表管理对象
 _Workbook book; //工作簿
 Worksheets sheets; //工作表管理对象
 _Worksheet sheet; //工作表管理
 Range2 range; //选区
 COleVariant
 covOptional((long)DISP_E_PARAMNOTFOUND, VT_ERROR);
 //创建Excel 2000服务器(启动Excel)
 if(!app.CreateDispatch("Excel.Application",NULL))
 {
 AfxMessageBox("创建Excel服务失败!");
 exit(1);
 }
 app.SetVisible(false); //不显示EXCEL
 //利用模板文件建立新文档
 CString ExcelPath = path;
 if(ExcelPath.Right(1) != "\\")
 ExcelPath += "\\";
 ExcelPath += "SQLToExcel"; //设置模板路径
 books.AttachDispatch(app.GetWorkbooks(),true);
 book.AttachDispatch(books.Add(_variant_t(ExcelPath)));
 sheets.AttachDispatch(book.GetWorksheets(),true); //得到Worksheets
 sheet.AttachDispatch(sheets.GetItem(_variant_t("sheet1")),true); //得到sheet1
 CString str1;
 str1 = "第1页";
 sheet.SetName(str1); //设置工作表
 for (int i=0;i<sheets.GetCount()-1;i++)
 {
 sheet = sheet.GetNext(); //获得下一页
 str1.Format("第%d页",i+2); //格式化字符串
 sheet.SetName(str1); //设置工作表
 }
 sheet.AttachDispatch(sheets.GetItem(_variant_t("第1页")),true); //得到第一页
 range.AttachDispatch(sheet.GetCells(),true); //得到全部Cells
 String sText[]={"字段名","字段类型","字段大小"}; //设置表头字符串
 for(int setnum=0;setnum<m_Frame.GetItemCount()+1;setnum++)
 {
 for(int num=0;num<3;num++)

```





```
{
if(!setnum) //判断是否插入表头
{
range.SetItem(_variant_t((long)(setnum+1)),_variant_t((long)(num+1)),
_variant_t(sText[num])); //插入表头
}
else
{
range.SetItem(_variant_t((long)(setnum+1)),_variant_t((long)(num+1)),
_variant_t(m_Frame.GetItemText(setnum-1,num))); //插入表文
}
}
}
sheet.SaveAs(strPath,covOptional,covOptional,covOptional,covOptional,
covOptional,covOptional,covOptional,covOptional);
app.SetVisible(true); //保存EXCEL表格
//显示EXCEL表格
//释放对象
range.ReleaseDispatch();
sheet.ReleaseDispatch();
sheets.ReleaseDispatch();
book.ReleaseDispatch();
books.ReleaseDispatch();
app.ReleaseDispatch();
}
```

### 举一反三

根据本实例，读者可以：

- 提取视图结构。

## 实例 397 加班网上管理

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\15\397

### 实例说明

公司工作特别繁忙的时候，需要加班加点工作，为了能快速并准确地统计加班人数，需要开发一个局域网加班软件。本实例为加班网上管理软件。运行程序，选择左侧列表中的图标，单击“加班”按钮进行加班登记，如果因临时有事而取消加班则双击右侧加班列表中的图标进行撤销，如图 15.3 所示。

### 技术要点

本实例技术要点为将员工姓名文件（employee.ini）和加班信息存储文件（overtime.ini）存放在服务器磁盘上，然后通过 WritePrivateProfileString 函数对 INI 文件进行输入。



图 15.3 加班网上管理

WritePrivateProfileString 函数向 INI 文件中写入指定节名和指定键名的字符串信息。语法：

```
BOOL WritePrivateProfileString(LPCTSTR lpAppName, LPCTSTR lpKeyName,
LPCTSTR lpString, LPCTSTR lpFileName);
```

参数说明：

- lpAppName：即将写入的节名。
- lpKeyName：即将写入节名下的键名。
- lpString：即将写入节名下键名的数据值。

- lpFileName: INI 文件名, 可以是全路径, 如果不是全路径, 默认就在系统文件夹下新建一个 INI 文件。
- 返回值: 如果成功, 返回非零值。

## 实现过程

- (1) 新建一个基于对话框的应用程序, 将窗体标题改为“加班网上管理”。
- (2) 向程序中导入 BMP 图片和 ICO 图标。
- (3) 向窗体中添加两个列表视图控件、一个图片控件和两个按钮控件。
- (4) 主要程序代码。

```

BOOL COverTimeDlg::PreTranslateMessage(MSG* pMsg)
{
 // TODO: Add your specialized code here and/or call the base class
 m_ToolTip.RelayEvent(pMsg);
 if(pMsg->message==WM_RBUTTONDOWN) //当用鼠标右键单击时弹出菜单
 {
 CMenu* pPopup = Menu.GetSubMenu(0); //获得子菜单
 CRect rc;
 CPoint point;
 GetCursorPos(&point); //获得鼠标位置
 rc.top=point.x;
 rc.left=point.y;
 pPopup->TrackPopupMenu(TPM_LEFTALIGN|TPM_LEFTBUTTON|TPM_VERTICAL,rc.top
 ,rc.left,this,&rc); //显示弹出菜单
 }
 if(pMsg->message==WM_LBUTTONDOWN) //如果按下鼠标左键
 {
 CRect rect,rc;
 m_ok.GetWindowRect(&rect); //获得按钮控件区域
 m_close.GetWindowRect(&rc);
 CPoint point;
 GetCursorPos(&point); //获得鼠标位置
 if(rect.PtInRect(point)) //判断是否在按钮区域内
 {
 CString str,name;
 char buf[128];
 int i=0,j;
 DWORD N;
 j = m_Gsyg.GetSelectionMark(); //获得当前选中列表项索引
 name = m_Gsyg.GetItemText(j,0); //获得列表项文本
 while(N!=7)
 {
 str.Format("%d",i+1);
 N = GetPrivateProfileString("加班员工",str,"default",buf,128,
 "///\\明日公共文件夹\\公共程序\\overtime.ini");//读取INI文件中数据
 if(buf == name)
 {
 MessageBox("你已经填写过加班信息!");
 return false;
 }
 }
 if(N!=7)
 {
 i++;
 }
 }
 if(MessageBox(name+": 请确定你是否加班! ", "加班信使服务软件",MB_YESNO)==IDYES)
 {
 str.Format("%d",i+1);
 WritePrivateProfileString(_T("加班员工"),_T(str),_T(name),
 _T("///\\明日公共文件夹\\公共程序\\overtime.ini")); //向INI文件中写入数据
 m_Jbyg.InsertItem(i,name,i);
 }
 STR.Format("加班人数: %d",i+1);
 m_ToolTip.UpdateTipText(STR,GetDlgItem(IDC_LIST2)); //修改提示信息
 }
 else if(rc.PtInRect(point))
 {

```

```

CDialog::OnCancel();
}
}
return CDialog::PreTranslateMessage(pMsg);
}

void COvertimeDlg::OnMenuClear()
{
 if(MessageBox("请确定是否清空加班名单!", "加班信使服务软件", MB_YESNO) == IDYES)
 {
 DWORD nSize = MAX_COMPUTERNAME_LENGTH + 1;
 char Buffer[MAX_COMPUTERNAME_LENGTH + 1];
 GetComputerName(Buffer, &nSize);
 CString Name;
 Name = Buffer;
 if(Name == "MRLRN") //判断用户权限
 {
 CFile file;
 //打开文件
 file.Open("\\\\明日公共文件夹\\公共程序\\overtime.ini", CFile::modeCreate | CFile::modeWrite);
 CString str;
 str = "[加班员工]";
 file.Write(str, str.GetLength()); //写入数据
 file.Close(); //关闭文件
 m_Jbyg.DeleteAllItems(); //删除控件中数据
 }
 else
 {
 MessageBox("对不起, 你没有足够权限清除加班信息!", "注意");
 return;
 }
 STR.Format("加班人数: %d", 0);
 m_ToolTip.UpdateTipText(STR, GetDlgItem(IDC_LIST2)); //修改提示信息
 }
}

//双击右侧列表中图标取消加班
void COvertimeDlg::OnDblclkList2(NMHDR* pNMHDR, LRESULT* pResult)
{
 CString str, name;
 int i, j;
 j = m_Jbyg.GetSelectionMark(); //获得列表中选择列表项索引
 name = m_Jbyg.GetItemText(j, 0); //获得列表项文本
 if(MessageBox(name + ": 请确定是否取消加班!", "加班信使服务软件", MB_YESNO) == IDYES)
 {
 m_Jbyg.DeleteItem(j);
 CFile file;
 file.Open("\\\\明日公共文件夹\\公共程序\\overtime.ini", CFile::modeCreate | CFile::modeWrite);
 CString str;
 str = "[加班员工]";
 file.Write(str, str.GetLength());
 file.Close();
 for(i = 0; i < m_Jbyg.GetItemCount(); i++)
 {
 str.Format("%d", i + 1);
 name = m_Jbyg.GetItemText(i, 0);
 WritePrivateProfileString(_T("加班员工"), _T(str), _T(name),
 _T("\\\\明日公共文件夹\\公共程序\\overtime.ini")); //向INI文件中写入数据
 }
 STR.Format("加班人数: %d", i);
 m_ToolTip.UpdateTipText(STR, GetDlgItem(IDC_LIST2));
 *pResult = 0;
 }
}

```

## 举一反三

根据本实例, 读者可以:

- 开发网络考勤管理。

## 实例 398 垃圾文件清理工具

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\15\398

## 实例说明

当 Windows 操作系统运行时会在磁盘中生成许多的临时文件，而这些临时文件在使用后没有及时删除就会形式垃圾文件。随着时间的推移产生的垃圾文件就越来越多，同时也占用了许多的磁盘空间。为了减少垃圾文件所占用的磁盘空间，可以使用本实例对这些垃圾文件进行清理。如图 15.4 所示。

## 技术要点

在该例子中主要是通过 CFileFind 对象查找磁盘中的垃圾文件。CFileFind 对象的方法有很多，主要有 FindFile、FindNextFile、IsDots、IsDirectory 和 GetFileName。下面介绍这几个方法。

(1) FindFile 方法。FindFile 方法用来查找指定文件。其语法格式如下：

```
virtual BOOL FindFile(LPCTSTR pstrName = NULL, DWORD dwUnused = 0);
```

参数说明：

- pstrName：指向文件名的字符串指针。
- dwUnused：固定值，该值为零。

- (2) FindNextFile 方法用于查找下一个文件，通过返回值可以判断是否是要查找的文件。
- (3) IsDots 方法判断目标文件是否是“.”或“..”。
- (4) IsDirectory 方法判断目标文件是否是文件夹。
- (5) GetFileName 方法获得查找到文件的文件名。

## 实现过程

(1) 新建一个基于对话框的应用程序。

(2) 在窗体上添加一个列表框控件，用来显示查找的垃圾文件。添加一个组合框控件，用来显示盘符。添加几个复选框控件，用来选择所要清理的垃圾文件的扩展名。

(3) 在窗体中单击“开始扫描”按钮，获取所要扫描的垃圾文件的扩展名，通过线程开始扫描，并将扫描结果显示在列表框控件中。实现代码如下：

```
void CClearGarbageDlg::OnBeginScan()
{
 if (m_bFinding == FALSE)
 {
 m_ScanInfo.DeleteAllItems();
 GetTmpExtendedName();
 m_bStopFind = FALSE;
 m_Disk.GetWindowText(m_szScanDisk);
 m_FindProgress.SetWindowText("查找进行中...");
 m_hThread = CreateThread(NULL, 0, ThreadProc, this, 0, NULL);
 }
}
```



图 15.4 垃圾文件清理工具



(4) 在窗体类中 GetTmpExtendedName 方法用来将窗体中选择的要清理的垃圾文件的扩展名添加到指定名表中。实现代码如下:

```
//获取临时文件扩展名
void CClearGarbageDlg::GetTmpExtendedName()
{
 //移除所有内容
 m_FilterList.RemoveAll();
 int nCheckID = IDC_TMP1;
 CButton* pBtn = NULL;
 CString szText;
 int nState = 0;
 for(int i= 0; i<20; i++, nCheckID++)
 {
 pBtn = (CButton*)GetDlgItem(nCheckID);
 if (pBtn != NULL)
 {
 nState = pBtn->GetCheck();
 if (nState)
 {
 pBtn->GetWindowText(szText);
 m_FilterList.AddTail(szText);
 }
 }
 }
}
```

(5) ThreadProc 函数为线程运行所需的线程函数, 该函数的作用就是查找指定扩展名的垃圾文件并添加到列表控件中。实现代码如下:

```
DWORD __stdcall ThreadProc(LPVOID lpParameter)
{
 CClearGarbageDlg* pDlg = (CClearGarbageDlg*) lpParameter;
 pDlg->m_bFinding = TRUE;
 pDlg->ResearchFile(pDlg->m_szScaneDisk);
 pDlg->m_FindProgress.SetWindowText("查找结束!");
 pDlg->m_bFinding = FALSE;

 return 0;
}

//查找文件
void CClearGarbageDlg::ResearchFile(CString szPath)
{
 CString strtemp;
 if(szPath.Right(1)!="\\")
 strtemp.Format("%s*.*",szPath);
 else
 strtemp.Format("%s*.*",szPath);
 CFileFind findfile;
 BOOL bfind=findfile.FindFile(strtemp);
 CString szRetName;
 while(bfind)
 {
 if (m_bStopFind) //结束查找
 {
 return;
 }
 bfind=findfile.FindNextFile();
 szRetName = findfile.GetFileName();
 if (IsTmpFile(szRetName))
 {
 CString szFullName = findfile.GetFilePath();
 int nCount = m_ScaneInfo.GetItemCount();
 }
 }
}
```

```

int nIndex = m_ScanInfo.InsertItem(nCount, "");
m_ScanInfo.SetItemText(nIndex, 0, szFullName);

}
if(findfile.IsDirectory() && !findfile.IsDots())
{
 ResearchFile(findfile.GetFilePath());
}
}
}

```

(6) 在窗体中单击“全部删除”按钮将会根据查找到的垃圾文件列表中的文件路径进行删除操作，当有文件不能删除时，将此文件路径显示在列表框控件中。实现代码如下：

```

void CClearGarbageDlg::OnDeleteAll()
{
 if (m_bFinding)
 {
 MessageBox("查找进行中，不能删除文件!");
 return;
 }
 if (MessageBox("确实要删除所有文件吗?", "提示", MB_YESNO) == IDYES)
 {
 CString szFileName = "";
 int nFileCount = m_ScanInfo.GetItemCount();
 m_DeleteLog.ResetContent();
 for (int i=0; i<nFileCount; i++)
 {
 szFileName = m_ScanInfo.GetItemText(i, 0);
 if (!DeleteFile(szFileName))
 {
 m_DeleteLog.AddString(szFileName + "文件删除失败!");
 }
 }
 }
}

```

### 举一反三

根据本实例，读者可以：

- 分类清理磁盘文件。

## 实例 399 网页照相机

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\15\399

### 实例说明

网页照相机是指将已打开的网页中的内容以图像的形式存储到图片文件中。本实例在窗体中放置了一个浏览器控件，这个控件一直是被隐藏着的。通过输入的 URL 地址在该浏览器控件中打开网页，再通过浏览器控件中提供的类方法获取网页快照。如图 15.5 所示。

### 技术要点

该实例使用的浏览器控件是一个标准的 ActiveX 控件，在该控件中提供的类中实现了 IHTMLDocument2 接口，通过该接口可以获取浏览器控件的大小和客户区。再通过 IViewObject 接口实现绘图操作，就可以将浏览器控件中显示的内容绘制到图片文件中。

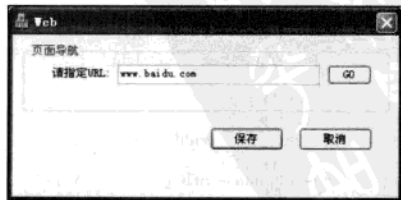


图 15.5 网页照相机

## 实现过程

- (1) 新建一个基于对话框的应用程序。
- (2) 在窗体上添加一个编辑框控件和两个按钮控件。
- (3) 当在编辑框控件中输入 URL 地址后单击“GO”按钮，将在浏览器控件中打开 URL

地址指定的页面。实现代码如下：

```
void CWebDlg::OnBnClickedLocate()
{
 CString szUrl;
 m_Url.GetWindowText(szUrl);
 if (!szUrl.IsEmpty())
 {
 m_Web.Navigate(szUrl, NULL, NULL, NULL, NULL);
 m_Web.MoveWindow(0, 0, 1, 1);
 }
}
```

(4) 当页面加载成功后单击“保存”按钮将弹出文件保存对话框，选择保存的文件路径后将网页中的内容绘制到图片文件中。实现代码如下：

```
void CWebDlg::OnBnClickedOk()
{
 IViewObject* pViewObject = NULL;
 LPDISPATCH pDocument = m_Web.get_Document();
 if (pDocument != NULL)
 {
 pDocument->QueryInterface(IID_IViewObject, (void**)&pViewObject);
 if (pViewObject != NULL)
 {
 IHTMLDocument2* pDocument2 = NULL;
 m_Web.get_Document()->QueryInterface(IID_IHTMLDocument2, (void**)&pDocument2);
 IHTMLElement* pElement = NULL;

 pDocument2->get_body(&pElement);
 IHTMLElement2* pScroll = NULL;

 pElement->QueryInterface(IID_IHTMLElement2, (void**)&pScroll);
 //获取页面的高度和宽度
 long nHeight, nWidth;
 pScroll->get_scrollHeight(&nHeight);
 pScroll->get_scrollWidth(&nWidth);
 //获取控件客户区域
 CRect clientRC;
 m_Web.GetClientRect(clientRC);

 int nBarWidth = GetSystemMetrics(SM_CYHSCROLL) + 5;
 CRect wndRC;
 GetClientRect(wndRC);
 m_Web.MoveWindow(wndRC.right, 0, nWidth + nBarWidth, nHeight + nBarWidth);

 BOOL bVerBar = FALSE;
 BOOL bHorBar = FALSE;

 pScroll->get_scrollHeight(&nHeight);
 pScroll->get_scrollWidth(&nWidth);

 //定义位图宽度和高度
 int nBmpWidth = nWidth;
 int nBmpHeight = nHeight;

 RECTL rect;
 rect.left = 0;
 rect.right = nWidth + nBarWidth;
 rect.top = 0;
 rect.bottom = nHeight + nBarWidth;

 CDC* pDC = GetDC();
 CDC srcDC, memDC;
 memDC.CreateCompatibleDC(pDC);
 srcDC.CreateCompatibleDC(pDC);
```

```

CBitmap srcBmp, bmp;
bmp.CreateCompatibleBitmap(pDC, nWidth, nHeight);

srcBmp.CreateCompatibleBitmap(pDC, nWidth + nBarWidth, nHeight + nBarWidth);

srcDC.SelectObject(&srcBmp);
memDC.SelectObject(&bmp);

pViewObject->Draw(DVASPECT_CONTENT, 1, 0, NULL,
0, srcDC.m_hDC, &rect, 0, NULL, 0);

memDC.BitBlt(0, 0, nWidth, nHeight, &srcDC,
1, 1, SRCCOPY);

//保存位图文件
int nAlign = 0;
nAlign = nBmpWidth % 4;
if (nAlign != 0)
 nAlign = 4 - nAlign;
//定义位图文件头
BITMAPFILEHEADER bmpFileHeader;
bmpFileHeader.bfReserved1 = 0;
bmpFileHeader.bfReserved2 = 0;
bmpFileHeader.bfSize = sizeof(BITMAPFILEHEADER);
bmpFileHeader.bfType = 0x4d42;
bmpFileHeader.bfOffBits = 54;
//定义位图信息头
BITMAPINFOHEADER bmpInfoHeader;
memset(&bmpInfoHeader, 0, sizeof(BITMAPINFOHEADER));
bmpInfoHeader.biHeight = nBmpHeight;
bmpInfoHeader.biWidth = nBmpWidth;
bmpInfoHeader.biPlanes = 1;
bmpInfoHeader.biBitCount = 24; //真彩色位图
bmpInfoHeader.biSizeImage = (nBmpHeight)*(nBmpWidth*3 + nAlign);
bmpInfoHeader.biCompression = 0;
bmpInfoHeader.biSize = sizeof(BITMAPINFOHEADER);

int nImageSize = nBmpHeight*(nBmpWidth*3 + nAlign);
BYTE* pBmpData = new BYTE[nImageSize];
memset(pBmpData, 255, nImageSize);
GetDIBits(memDC.m_hDC, bmp, 0, nBmpHeight, pBmpData,
(BITMAPINFO*)&bmpInfoHeader,
DIB_RGB_COLORS);

CFileDialog fDlg(FALSE, L"bmp", L"web.bmp", OFN_HIDEREADONLY |
 OFN_OVERWRITEPROMPT);
if (fDlg.DoModal() == IDOK)
{
 CString szFileName = fDlg.GetPathName();
 CFile file;
 file.Open(szFileName, CFile::modeCreate|CFile::modeReadWrite);
 file.Write(&bmpFileHeader, sizeof(BITMAPFILEHEADER));
 file.Write(&bmpInfoHeader, sizeof(BITMAPINFOHEADER));
 file.Write(pBmpData, nImageSize);
 file.Close();
}
delete [] pBmpData;
bmp.DeleteObject();
srcDC.DeleteDC();
memDC.DeleteDC();
}
}

```

## 举一反三

根据本实例，读者可以：

- 获取网页源码。



## 实例 400 屏幕截图工具

本实例是一个提高效率、人性化的程序

实例位置：光盘\mingrisoft\15\400

## 实例说明

用户在日常的工作中，有时需要将电脑屏幕保存为图片，可是使用键盘上的<PrtScSysRq>键却不能抓取屏幕上的鼠标，本实例通过 Visual C++ 实现了屏幕截图的功能，并且允许用户在截图的过程中抓取鼠标。运行本实例，单击“...”按钮，在弹出的“浏览文件夹”对话框中选择图片存储路径，并选择是否抓取鼠标，设置抓屏热键，单击“保存”按钮保存设置，在需要截图时直接按下设置的热键就可以将抓取的图片保存到设置的存储路径下。如图 15.6 所示。

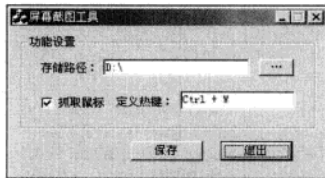


图 15.6 屏幕截图工具

## 技术要点

本示例中实现屏幕截图功能时，主要用 RegisterHotKey 函数、UnregisterHotKey 函数、GetDesktopWindow 函数和 CFile 类，下面对本实例中用到的关键技术进行详细讲解。

(1) RegisterHotKey 函数。RegisterHotKey 函数用于注册系统热键，该函数的语法格式如下：

```
BOOL RegisterHotKey(HWND hWnd, int id, UINT fsModifiers, UINT vk);
```

参数说明：

- hWnd：接收热键产生 WM\_HOTKEY 消息的窗口句柄。若该参数 NULL，传递给调用线程的 WM\_HOTKEY 消息必须在消息循环中进行处理。
- id：定义热键的标识符。
- fsModifiers：定义为了产生 WM\_HOTKEY 消息而必须与由 nVirtKey 参数定义的键一起按下的键。
- vk：定义热键的虚拟键码。

本实例中用于注册系统热键的代码如下：

```
m_HotKey.SetHotKey(wvk,wmod); //设置热键控件中的键值
BOOL result = RegisterHotKey(this->GetSafeHwnd(),HOTKEY_GRASP,wmod,wvk); //注册系统热键
if(!result) //判断是否注册成功
{
 MessageBox("注册热键失败");
}
```

(2) UnregisterHotKey 函数。UnregisterHotKey 函数用于释放调用线程先前登记的热键，其语法格式如下：

```
BOOL UnregisterHotKey(HWND hWnd, int id);
```

参数说明：

- hWnd：与被释放的热键相关的窗口句柄。若热键不与窗口相关，则该参数为 NULL。
- id：定义被释放的热键的标识符。

(3) GetDesktopWindow 函数。GetDesktopWindow 函数用于获取系统桌面窗口，返回值是系统桌面窗口指针，通过该指针可以操作 Windows 桌面，其语法格式如下：

```
static CWnd* PASCAL GetDesktopWindow();
```

(4) CFile 类的 Open 方法。Open 方法用于打开一个文件，它可以和没有参数的构造函数 CFile 一起使用，用来以安全的方式打开文件，其语法格式如下：

```
virtual BOOL Open(LPCTSTR lpszFileName, UINT nOpenFlags, CFileException* pError = NULL);
```

参数说明:

- `IpszFileName`: 将要打开的文件的路径, 可以是绝对路径和相对路径, 网络路径也可以。
- `nOpenFlags`: 一个定义了文件的共享和访问模式的 `UINT`。它指定了打开文件后的动作, 可以用 `OR (|)` 操作符将选项组合起来, 至少应有一个访问权限和一个共享选项, `modeCreate` 和 `modeNoInherit` 模式是可选的。
- `pError`: 一个异常的指针, 一般情况下可以使用 `NULL` 指针, 这个指针在打开文件过程中如果产生错误, `Open` 将抛出一个 `CFileException` 异常, 而不是返回 `FALSE`。

(5) `CFile` 类的 `WriteHuge` 方法。`WriteHuge` 方法用于将缓冲区中的数据写入到文件, 主要用来写入比较大的数据, 其语法格式如下:

```
void WriteHuge(const void* lpBuf, DWORD dwCount);
```

参数说明:

- `lpBuf`: 提供将要写入的数据的缓冲区。
- `dwCount`: 将要写入数据的大小。

(6) `CFile` 类的 `Close` 方法。`Close` 方法用于关闭 `CFile` 对象, 其语法格式如下:

```
virtual void Close();
```

## 实现过程

(1) 新建一个基于对话框的应用程序, 将其窗体标题改为“屏幕截图工具”, 勾选 `Minimize box` 属性, 使对话框具有最小化按钮。

(2) 向对话框中添加 1 个群组控件、2 个静态文本控件、1 个编辑框控件、1 个复选框控件、1 个热键控件和 3 个按钮控件。

(3) 引用 `windowsx.h` 头文件和 `math.h` 头文件并声明常量, 代码如下:

```
#include <windowsx.h>
#include "math.h"
#define HOTKEY_GRASP 10000
```

(4) 在主窗体初始化时读取 `INI` 文件中数据, 根据读取的数据设置控件默认显示, 并注册系统热键, 代码如下:

```
m_Num = 1;
WORD wvk,wmod;
char bufwvk[8],bufwmod[8],bufpath[256],bufmouse[2];
GetPrivateProfileString("Set","path","",bufpath,256,"./System.ini"); //获得存储路径
m_Path.SetWindowText(bufpath); //显示存储路径
GetPrivateProfileString("Set","wvk","",bufwvk,8,"./System.ini"); //热键的虚拟代码
wvk = atoi(bufwvk); //转换为整数
GetPrivateProfileString("Set","wmod","",bufwmod,8,"./System.ini"); //获得修正标志
wmod = atoi(bufwmod); //转换为整数
GetPrivateProfileString("Set","capmouse","",bufmouse,2,"./System.ini"); //获得是否捕捉鼠标标识
m_Cursor.SetCheck(atoi(bufmouse)); //设置复选框默认值
m_HotKey.SetHotKey(wvk,wmod); //设置热键控件默认值
BOOL result = RegisterHotKey(this->GetSafeHwnd(),HOTKEY_GRASP,wmod,wvk); //注册系统热键
if(!result) //判断热键是否注册成功
{
 MessageBox("注册热键失败");
}
```

(5) 处理“...”按钮的单击事件, 在该事件中调用文件浏览对话框设置图片的存储路径, 其实现代码如下:

```
void CGraspBmpDlg::OnButpath()
{
 // TODO: Add your control notification handler code here
 CString ReturnPath; //字符串变量
 TCHAR szPath[_MAX_PATH]; //保存路径变量
 BROWSEINFO bi; //BROWSEINFO结构变量
```

```

bi.hwndOwner = NULL; //HWND句柄
bi.pidlRoot = NULL; //默认值为NULL
bi.lpszTitle = _T("文件浏览对话框"); //对话框标题
bi.pszDisplayName = szPath; //选择文件夹路径
bi.ulFlags = BIF_RETURNONLYFSDIRS; //标记
bi.lpfm = NULL; //默认值为NULL
bi.lParam = NULL; //回调消息
LPITEMIDLIST pltemIDList = SHBrowseForFolder(&bi); //显示文件浏览对话框
if(pltemIDList)
{
 if(SHGetPathFromIDList(pltemIDList,szPath)) //判断是否获得文件夹路径
 ReturnPach = szPath; //获得文件夹路径
 else
 {
 ReturnPach = ""; //文件夹路径为空
 }
 m_Path.SetWindowText(ReturnPach); //显示存储路径
}

```

(6) 手动添加 WM\_HOTKEY 消息的处理函数, 在该函数中调用 StartGrasp 函数抓取屏幕并生成 BMP 位图文件, 实现代码如下:

```

void CGraspBmpDlg::OnHotKey(WPARAM wParam,LPARAM lParam)
{
 if(HOTKEY_GRASP == (int)wParam)
 {
 StartGrasp(); //抓图
 }
}

```

(7) 添加自定义函数 StartGrasp, 该函数用于进行屏幕抓图, 其实现代码如下:

```

void CGraspBmpDlg::StartGrasp()
{
 CString path;
 m_Path.GetWindowText(path); //获得文件路径
 if(path.IsEmpty()) //判断文件路径是否为空
 {
 MessageBox("请选择文件存储位置");
 return;
 }
 CDC* pDeskDC = GetDesktopWindow()->GetDC(); //获取桌面画布对象
 CRect rc;
 GetDesktopWindow()->GetClientRect(rc); //获取屏幕的客户区域
 CDC memDC; //定义一个内存画布
 memDC.CreateCompatibleDC(pDeskDC); //创建一个兼容的画布
 CBitmap bmp;
 bmp.CreateCompatibleBitmap(pDeskDC,rc.Width(),rc.Height()); //创建兼容位图
 memDC.SelectObject(&bmp); //选中位图对象
 BITMAP bitmap;
 bmp.GetBitmap(&bitmap); //获得图片信息
 memDC.BitBlt(0,0,bitmap.bmWidth,bitmap.bmHeight,pDeskDC,0,0,SRCCOPY); //绘制图片
 if(m_Cursor.GetCheck()) //选择抓取鼠标
 {
 CPoint point;
 GetCursorPos(&point); //鼠标位置
 HICON hicon = (HICON)GetCursor(); //获得鼠标图标句柄
 memDC.DrawIcon(point.x-10,point.y-10,hicon); //绘制鼠标图标
 }
 DWORD size=bitmap.bmWidthBytes*bitmap.bmHeight; //图片数据大小
 LPSTR lpData=(LPSTR)GlobalAllocPtr(GPTR,size);
 int panelSize = 0; //记录调色板大小
 if(bitmap.bmBitsPixel<16) //判断是否为真彩色位图
 {
 panelSize = pow(2,bitmap.bmBitsPixel*sizeof(RGBQUAD));
 BITMAPINFOHEADER *pBInfo = (BITMAPINFOHEADER*)LocalAlloc(LPTR,
 sizeof(BITMAPINFO)+panelSize); //位图头指针
 pBInfo->biBitCount = bitmap.bmBitsPixel; //位图像素
 }
}

```

```

pBInfo->biClrImportant = 0;
pBInfo->biCompression = 0;
pBInfo->biHeight = bitmap.bmHeight; //位图高
pBInfo->biPlanes = bitmap.bmPlanes;
pBInfo->biSize = sizeof(BITMAPINFO);
pBInfo->biSizeImage = bitmap.bmWidthBytes*bitmap.bmHeight; //数据
pBInfo->biWidth = bitmap.bmWidth; //位图宽
pBInfo->biXPelsPerMeter = 0;
pBInfo->biYPelsPerMeter = 0;
GetDIBits(memDC.m_hDC,bmp,0,pBInfo->biHeight,lpData,
(BITMAPINFO*)pBInfo,DIB_RGB_COLORS);
CString name,str;
CTime time = CTime::GetCurrentTime(); //抓图时间
str.Format("%04d",m_Num++);
if(path.Right(1) == "\\")
name = path+time.Format("%Y%m%d")+str+".bmp"; //设置文件名及路径
else
name = path+"\\")+time.Format("%Y%m%d")+str+".bmp"; //设置文件名及路径
BITMAPFILEHEADER bfh; //位图文件头
bfh.bfReserved1 = bfh.bfReserved2 = 0;
bfh.bfType = ((WORD)('M'<<8) | 'B');
bfh.bfSize = 54+size;
bfh.bfOffBits = 54;
CFile file;
if(file.Open(name,CFile::modeCreate|CFile::modeWrite)) //创建位图文件
{
file.WriteHuge(&bfh,sizeof(BITMAPFILEHEADER)); //写入位图文件头
file.WriteHuge(pBInfo,sizeof(BITMAPINFOHEADER)); //写入文件头
file.WriteHuge(lpData,size); //写入数据
file.Close(); //关闭文件
}
}

```

(8) 处理“保存”按钮的单击事件，在该事件的处理函数中获得用户设置的信息，并将信息保存到 INI 文件中，其实现代码如下：

```

void CGraspBmpDlg::OnButsave()
{
::UnregisterHotKey(this->GetSafeHwnd(),HOTKEY_GRASP); //释放注册的热键
WORD wvk,wmod;
m_HotKey.GetHotKey(wvk,wmod); //获得热键控件中的键值
BOOL result=RegisterHotKey(this->GetSafeHwnd(),HOTKEY_GRASP,wmod,wvk); //注册热键
if(!result)
{
MessageBox("注册热键失败");
return;
}
CString path,strwvk,strwmod,strmouse;
m_Path.GetWindowText(path); //获得存储路径
WritePrivateProfileString("Set","path",path,"./System.ini"); //记录存储路径
strwvk.Format("%d",wvk); //获得虚拟键值
WritePrivateProfileString("Set","wvk",strwvk,"./System.ini"); //记录虚拟键值
strwmod.Format("%d",wmod); //获得修改标识
WritePrivateProfileString("Set","wmod",strwmod,"./System.ini"); //记录修改标识
strmouse.Format("%d",m_Cursor.GetCheck()); //获得抓取鼠标标识
WritePrivateProfileString("Set","capmouse",strmouse,"./System.ini"); //记录鼠标标识
}

```

## 举一反三

根据本实例，读者可以：

- 图像的剪切和合成。



蘇子知覺

PDG

## 附录 技术要点对应实例位置

带\*号的实例表示在对应实例的“技术要点”部分对指定技术有较详细的介绍。不带\*号的实例表明在对应实例的“实现过程/主要代码”部分使用了该技术。

名 称	实 例	名 称	实 例
<b>A</b>		CreateFileMapping 函数	实例 358*
AbortPath 方法	实例 135*	CreateFile 函数	实例 180*、实例 321*
AddString 方法	实例 067*	CreateFont 方法	实例 132*、实例 167
AddTool 方法	实例 097*	CreateProcess 函数	实例 206*、实例 235*、实例 236*
AdjustTokenPrivileges 函数	实例 209*	CreateThread 函数	实例 244*、实例 371
<b>B</b>		CreateToolhelp32Snapshot 函数	实例 232*
Beep 函数	实例 163*	CreateDragImage 方法	实例 077*、实例 082*
BeginPaint 方法	实例 095*	CreateEllipticRgn 方法	实例 127*
BeginPath 方法	实例 042*、实例 134*、实例 135*	CreateEvent 函数	实例 240*
BitBlt 方法	实例 086*	CreateMutex 函数	实例 239*
Biokey 组件	实例 348*	CreatePolygonRgn 方法	实例 127*
<b>C</b>		CreateRectRgn 方法	实例 127*
CCheckBox 类	实例 063*	CreateSemaphore 函数	实例 243*
CDC 类的 GetDeviceCaps 方法	实例 307*	CreateService 函数	实例 238*
CFileFind 类	实例 177*、实例 398*	CreateStatic 方法	实例 028*
CHtmlView 类	实例 378*	CreateStreamOnHGlobal 函数	实例 123*
CHttpFile 类	实例 382*、实例 383*	CreateView 方法	实例 028*
CInternetSession 类	实例 370*、实例 382*	CREATE VIEW 语句	实例 272*
ClipCursor 函数	实例 246*	CRichEdit 类方法	实例 089
CombineRgn 函数	实例 022*、实例 127*	CSocket 类	实例 361*
CopyCursor 函数	实例 245*	<b>D</b>	
CopyFile 函数	实例 187*	DeleteFile 方法	实例 188*
Cpackage 类	实例 386*	Delete 语句	实例 270*、实例 271*
CreateDIBSection 函数	实例 115*	DisplayPath 函数	实例 074*
CreateDirectory 函数	实例 173*、实例 182*	DllRegisterServer 函数	实例 225*

续表

名 称	实 例	名 称	实 例
DllUnregisterServer 函数	实例 225*	G	
Draw3dRect 方法	实例 143	GetAt	实例 387*
DrawDib 库	实例 115*、实例 116*、 实例 117*、实例 118*	GetCheck 函数	实例 083*
DrawDibDraw 函数	实例 115*、实例 116*	GetClassName 函数	实例 314*
DrawItem 方法	实例 002*、实例 006*、 实例 051、实例 070*、 实例 071*、实例 078*	GetClipboardData 函数	实例 230*
DrawText 方法	实例 101*	GetCommandLine 方法	实例 236*
DROP VIEW 语句	实例 274*	GetCurrentTime 函数	实例 025*
E		GetCurSel 方法	实例 100*
Ellipse 方法	实例 107*	GetCursor 函数	实例 245*
EN_CHANGE 消息	实例 055*	GetCursorPos 函数	实例 053*
EndPaint 方法	实例 095*	GetDefaultCharFormat 方法	实例 092*
EndPath 方法	实例 042*、实例 134*、 实例 135*	GetDesktopWindow 函数	实例 400*
EnterCriticalSection 函数	实例 242*	GetDIBits 函数	实例 315*
EnumClipboardFormats 函数	实例 230*	GetDiskFreeSpaceEx 函数	实例 198*、实例 212*
EnumFontFamiliesEx 函数	实例 133*	GetDriveType 函数	实例 211*
EnumWindows 函数	实例 234*	gethostbyname 函数	实例 351*、实例 367*
ExitWindowsEx 函数	实例 209*	GetItemRect 方法	实例 066*
ExtractIcon 函数	实例 139*	GetKeyboardState 函数	实例 249*
F		GetLogicalDriveStrings 函数	实例 073*、实例 087*、 实例 138*、实例 198*、 实例 212*
FillRect 方法	实例 096*	GetNextItem 方法	实例 081*
FillSolidRect 方法	实例 101*	GetOpenFileName 函数	实例 045*
FindFile 方法	实例 197*	GetPixel 方法	实例 119*、实例 120*、 实例 122*、实例 130*
FindFirstFile 函数	实例 138*	GetPos 方法	实例 098*
FindNextFile 函数	实例 138*、实例 197*	GetprivateProfileSection	实例 003
FindResource 函数	实例 050*	GetPrivateProfileString 函数	实例 391*
FindWindow 函数	实例 216*、实例 217*、 实例 218*	GetPrivateProfileStruct 函数	实例 203*
FindWindowEx 函数	实例 216*、实例 217*、 实例 218*	GetProcAddress 函数	实例 215*
FlashWindow 函数	实例 038*	GetRValue 宏	实例 129*
FormatDriver 函数	实例 215*	GetSelectionMark 方法	实例 076*
FrameRect 函数	实例 056*	GetPrivateProfileSectionNames	实例 003

续表

名 称	实 例	名 称	实 例
GetStatus 方法	实例 190*	LookupPrivilegeValue 函数	实例 209*
GetSystemDirectory 函数	实例 221*	M	
GetSystemMenu 方法	实例 001*、实例 040*	MAPI 函数	实例 375*
GetSystemMetrics 函数	实例 069*、实例 140*、 实例 165*、实例 228*	MapViewOfFile 函数	实例 358*
GetText 方法	实例 066*	MapWindowPoints 函数	实例 037*
GetToolBarCtrl 方法	实例 009*、实例 013*	MciSendCommand 函数	实例 158*
GetTopIndex 方法	实例 066*	MCIWndCreate 函数	实例 160*
GetVolumeInformation 函数	实例 213*	MCIWndNew 函数	实例 160*
GetWindowsDirectory 函数	实例 221*	MCIWndCanRecord 函数	实例 160*
GlobalMemoryStatus 函数	实例 229*	MeasureItem 方法	实例 002*、实例 006*
GlobalAlloc 函数	实例 123*	MoveFile 方法	实例 183*、实例 191*
GlobalLock 函数	实例 123*	MoveTo 和 LineTo 函数	实例 104*、实例 312*、 实例 313*
GraphEdit 工具	实例 168*、实例 170*	MoveWindow 函数	实例 016、实例 037*、 实例 079、实例 142、 实例 165*
I		MSComm 控件	实例 322*、实例 339*、 实例 365*
IC 卡函数	实例 327*、实例 328*、 实例 329*	N	
IFS 算法	实例 109*	Netbios 函数	实例 352*、实例 393*
ImgScan 控件	实例 338*	NtQuerySystemInformation 函数	实例 176*
InitializeCriticalSection 函数	实例 242*	O	
InsertItem 方法	实例 080*、实例 081*	OCX 控件 CGIF	实例 125*
Invalidiate 函数	实例 131	OCX 控件 CJPG	实例 124*
InvertRgn 方法	实例 121*	OleLoadPicture 函数	实例 123*
IpropertySetStorage 接口	实例 194*	OnCtlColor 消息	实例 057*
IREALmagicCtrl 控件	实例 347*	OnHScroll 消息	实例 112*
L		OnToolbarDropDown 方法	实例 004*
LeaveCriticalSection 函数	实例 242*	open 函数	实例 032*、实例 049*、 实例 179*
LineFromChar 方法	实例 090*	OpenClipboard 函数	实例 230*
LineMakeCall 函数	实例 346*	OpenGL 库	实例 148*、实例 149、 实例 150
LoadBitmap 函数	实例 250*	OpenProcessToken 函数	实例 209*
LoadCursorFromFile 函数	实例 245*	P	
LoadImage 函数	实例 091*、实例 111*	PolyBezier 方法	实例 106*
LoadLibrary 函数	实例 215、实例 225*	PostMessage 函数	实例 357*
LoadResource 函数	实例 050*	PreTranslateMessage 方法	实例 033*、实例 034*



续表

名 称	实 例	名 称	实 例
Progress 控件	实例 185*	SetBkMode 方法	实例 059*、实例 101*
R		SetButtonInfo 方法	实例 011*
RasEnumConnections 函数	实例 369*	SetCurSel 方法	实例 100*
RasEnumEntries 函数	实例 364*	SetCursor 函数	实例 062*、实例 248*
Read、Write 函数	实例 200*、实例 204*	SetDelayTime 方法	实例 097*
ReadDirectoryChangesW 函数	实例 227*	SetEditSel 方法	实例 068*
ReadFile 函数	实例 321*	SetFileAttributes 函数	实例 187*
ReadString 方法	实例 192*、实例 193*	SendMessage 函数	实例 178*
RealPlayer 组件	实例 161*	SetDIBits 函数	实例 356*
Rectangle 方法	实例 107*、实例 108*	SetImageList 方法	实例 080*、实例 084*
RegCreateKey 函数	实例 208*、实例 210*、 实例 253*、实例 254*、 实例 255*、实例 256*、 实例 259*、实例 260*、 实例 261*、实例 262*、 实例 263*、实例 392*	SetItemText 方法	实例 075*
RegDeleteKey 函数	实例 208、实例 252*、 实例 257*	SetLayeredWindowAttributes 函数	实例 041*
RegisterHotKey 函数	实例 220*	SetLineColor 方法	实例 085*
RegisterWindowMessage 函数	实例 357*	SetMaxTipWidth 方法	实例 097*
RegQueryValueEx 函数	实例 392*	SetPixel 方法	实例 105*、实例 119、 实例 120*、实例 122*、 实例 130*
RegSetValueEx 函数	实例 252*、实例 253*、 实例 254*、实例 255*、 实例 256*、实例 259*、 实例 260*、实例 261*、 实例 262*、实例 263*	SetRange 方法	实例 098*、实例 185*
ReleaseMutex 函数	实例 239*	SetStatus 方法	实例 190*、实例 199*
RemoveDirectory 函数	实例 181*、实例 182*	SetTextColor 函数	实例 057*、实例 058、 实例 101*
rename 函数	实例 189*、实例 191*、 实例 201*	SetTimer 方法	实例 017、实例 110*、 实例 141*、实例 142、 实例 368*、实例 381*
ReplaceSel 方法	实例 092*	SetSystemCursor 函数	实例 245*
S		SetViewportOrg 方法	实例 104*、实例 108*
Select into 语句	实例 267*	SetWindowExt 函数	实例 320*
SelectObject 方法	实例 101*	SetWindowPos 函数	实例 035*、实例 088*、 实例 218*
SelectString 方法	实例 068*	SetWindowsHookEx 函数	实例 231*
SetAt	实例 387*、实例 389	SetWordCharFormat 方法	实例 092*
SetBitmap 方法	实例 021、实例 103*	Shell_NotifyIcon 函数	实例 007*

续表

名 称	实 例	名 称	实 例
ShellExecute 函数	实例 062*、实例 205*、 实例 223*、实例 316*、 实例 368*	WM_DEVICECHANGE 消息	实例 226*
SHBrowseForFolder 函数	实例 187*	WM_GETMINMAXINFO 消息	实例 027*
SHEmptyRecycleBin 函数	实例 175*	WM_KEYDOWN 消息	实例 335*
SHFileOperation 函数	实例 174*、实例 184*	WM_LBUTTONDOWN 事件	实例 020*、实例 033*、 实例 136*
SHGetFileInfo 函数	实例 187*	WM_LBUTTONUP 事件	实例 020*、实例 136*
SHGetSpecialFolder Location 函数	实例 233*	WM_MOUSEMOVE 消息	实例 020*、实例 021、 实例 136*、实例 166*
ShowWindow 方法	实例 102、实例 143	WNetAddConnection2 函数	实例 360*
SizeofResource 函数	实例 050*	WNetCloseEnum 函数	实例 350*、实例 354、 实例 359
StretchBlt 方法	实例 024*、实例 094、 实例 113*、实例 126*、 实例 128*、实例 143、 实例 309*、实例 310*、 实例 311、实例 381*	WNetEnumResource 函数	实例 350*、实例 354、 实例 359
StretchDIBits 函数	实例 316*	WNetOpenEnum 函数	实例 350*、实例 354、 实例 359
StrokePath 方法	实例 134*	WriteFile 函数	实例 321*
SystemParametersInfo 函数	实例 219*	WritePrivateProfileSection 函数	实例 202*
T		WritePrivateProfileString 函数	实例 031*、实例 180*、 实例 187*、实例 397*
TextOut 方法	实例 042*	WritePrivateProfileStruct 函数	实例 202*
TrackPopupMenu 方法	实例 005*、实例 054*	WSAEnumProtocols 函数	实例 366*
U		其 他	
UnregisterHotKey 函数	实例 220*	ADO 对象	实例 265*、实例 273
Update 语句 (SQL)	实例 268*、实例 269*	AES 算法	实例 390*
UpdateTipText 方法	实例 097*	AVI 函数库	实例 154*、实例 171*
W		DirectShow 技术	实例 172*
WaitForSingleObject 函数	实例 240*	DLL 技术	实例 237*、实例 251
WaveInOpen 函数	实例 362*	DirectX 技术	实例 349*
WaveInPrepareHeader 函数	实例 362*	ID 卡函数	实例 330*
WebBrowser 控件	实例 155*、实例 378*、 实例 380*	GDI+ 技术	实例 114*、实例 137*、 实例 151、实例 152
WinExec 函数	实例 214*、实例 223*、 实例 353*	JMail 组件	实例 373*、实例 374*
winmm.lib 库	实例 164*、实例 169、 实例 222、实例 363*	MD5 算法	实例 388*

续表

名 称	实 例	名 称	实 例
MSHTML 组件	实例 376*	ZQPntCtrl.dll 动态库	实例 336*
MSWORD9.OLB 库	实例 195*、实例 196*	打印对话框 CPrintDialog	实例 308*
msxml.dll 动态库	实例 207*、实例 379*	打印结构 DEVMODE	实例 317*
ODBC 数据源	实例 264*	短信猫动态库	实例 344*、实例 345*
PE 档案	实例 186*	加密狗硬件	实例 323、实例 324*
SHDocVw 组件	实例 376*、实例 384	加密锁硬件	实例 325*、实例 326*
TAPI 函数	实例 385*	语音卡动态库	实例 340*、实例 341*、 实例 342*、实例 343*
VFW 技术	实例 331*、实例 332*、 实例 333*、实例 334*、 实例 363*	设计 ATL 控件	实例 093*
Windows Socket API 函数	实例 355*		